GC26-3819-3
File No. S370-30

Systems

OS/VS Virtual Storage Access Method (VSAM) Options for Advanced Applications

VS1 Release 6
VS2 Release 3.7

PLEASE RETURN TO:
PANSOPHIC SYSTEM INC.
DATACENTER LIBRARY

IBM

Fourth Edition (January 1976)

This edition, as amended by technical newsletter GN26-0898, applies both to Release 6 of OS/VS1 and to Release 3.7 of OS/VS2, and to any subsequent releases of either system unless otherwise indicated in new editions or technical newsletters.

Significant system changes are summarized under "Summary of Amendments" following the list of figures. Each technical change is marked by a vertical line to the left of the change.

Information in this publication is subject to significant change. Any such changes will be published in new editions or technical newsletters. Before using the publication, consult the latest *IBM System/370 Bibliography,* GC20-0001, and the technical newsletters that amend the bibliography, to learn which editions and technical newsletters are applicable and current.

Requests for copies of IBM publications should be made to the IBM branch office that serves you.

Forms for readers' comments are provided at the back of the publication. If the forms have been removed, comments may be addressed to IBM Corporation, P.O. Box 50020, Programming Publishing, San Jose, California 95150. All comments and suggestions become the property of IBM.

© Copyright International Business Machines Corporation 1973, 1975, 1976

HOW TO USE THIS PUBLICATION

This publication describes applications of VSAM (Virtual Storage Access Method) that might be effective in the development of utility programs and system control programs.

The chapters of this publication are:

- "Summary of Amendments," which gives the changes to this publication for the releases of OS/VS1 and OS/VS2.
- "Processing Control Intervals," which describes how to gain access to the whole contents of a control interval (rather than to individual data records) and how to manage your own I/O (input/output) buffers with control-interval access.
- "Sharing Resources among Data Sets," which describes how to build a VSAM resource pool for sharing I/O buffers, I/O-related control blocks, and channel programs and how to manage buffers in a buffer pool.
- "Processing a VSAM Index," which gives the format of a record in the index of a key-sequenced data set and explains how to gain access to an index using either the GETIX and PUTIX macros or the GET and PUT macros.
- "Processing a VSAM Catalog," which describes how to gain access to a VSAM catalog as a catalog, by way of the SHOWCAT macro, and as a data set, by way of the GET and PUT macros.
- "Building Parameter Lists for GENCB, MODCB, SHOWCB, and TESTCB," which gives the format of the parameter lists used by GENCB, MODCB, SHOWCB, and TESTCB and explains how to build a parameter list to generate, modify, display, or test an access-method control block, exit list, or request parameter list.
- "Appendix: Operand Notation for the Macros," which describes list and execute forms of the macros presented in this publication and expressions that can be used to code the operands of each macro.

This publication also has:

- "Glossary," which defines technical terms used in this publication.
- "Index," which identifies the pages on which the subjects of this publication are discussed.

The formats of the control information in a control interval, of a VSAM index record, and of the macros for building parameter lists for GENCB, MODCB, SHOWCB, and TESTCB are subject to change. The publication of this information implies no commitment on the part of IBM to maintain these formats in future releases.

Conventions Used in the Publication

The conventions used in this publication for presenting the formats of macros indicate whether an operand is optional, how to specify the value for an operand, and how to punctuate a macro. The conventions are:

- Expressions enclosed in brackets, [ ], are optional.
- Items separated by an OR sign, |, are alternatives, only one of which may be specified. The alternatives are enclosed in braces, { }.
- An underlined **item** is the default when you don't specify anything for an operand.
- Ellipses, ..., indicate either that you may repeat the preceding item or that other items, not relevant to the discussion, are omitted.
- Capitalized **BOLD** expressions, parentheses, commas, and equal signs must be entered as shown, except that, unless otherwise noted, parentheses aren't required if you specify only one item.
- Lowercase *italic* expressions are variables for which you may specify one of a number of expressions.

Required Publications

The *OS/VS Virtual Storage Access Method (VSAM) Programmer's Guide,* GC26-3838, is recommended for learning about VSAM data sets and the macros you use to gain access to them. Rather than repeat information from the *VSAM Programmer's Guide*, this publication assumes you are familiar with the *VSAM Programmer's Guide.* This publication discusses the operands of some of the macros described in that publication and some of the return codes from those macros, but it relies on that publication for a full presentation of the macros and return codes.

Related Publications

The following publications are referred to for additional information about particular subjects:

OS/VS Publication

- *OS/VS1 Checkpoint/Restart,* GC26-3876, and *OS/VS2 MVS Checkpoint/Restart,* GC26-3877, for information about the use of checkpoint/restart with a VSAM data set.

OS/VS1 Publications

- *OS/VS1 Access Method Services,* GC26-3840, for information about the commands used to process a catalog as a catalog.
- *OS/VS1 Virtual Storage Access Method (VSAM) Logic,* SY26-3841, for information about the format and arrangement of records in a VSAM catalog.

OS/VS2 Publications

- *OS/VS2 Access Method Services,* GC26-3841, for information about the commands used to process a catalog as a catalog.
- *OS/VS2 Catalog Management Logic,* SY26-3826, for information about the format and arrangement of records in a VSAM catalog.

- *OS/VS2 System Programming Library: Supervisor*, GC28-0628, for information about the authorized program facility and service management (SRB dispatching).

CONTENTS

How to Use This Publication ... 3
Conventions Used in the Publication ... 4
Required Publications ... 4
Related Publications ... 4

Figures ... 9

Summary of Amendments ... 11
OS/VS1 Release 5 and OS/VS2 Release 3.7 ... 11
GENCB, MODCB, SHOWCB, and TESTCB ... 11
OS/VS1 Release 4 and OS/VS2 Independent Component ... 11
Control-Interval Access ... 11
Sharing Resources ... 11
Index Processing ... 11
Catalog Processing ... 11
OS/VS2 Release 3 ... 11

Processing Control Intervals ... 13
The Format of a Control Interval ... 13
CIDF—Control-Interval Definition Field ... 13
RDF—Record Definition Field ... 14
How to Gain Access to a Control Interval ... 17
Managing Your Own I/O Buffers ... 18
Improved Control-Interval Access ... 18
CBIC (Control Blocks in Common) Option (MVS only) ... 20
Error Return Codes ... 20

Sharing Resources Among Data Sets ... 21
Providing a Resource Pool ... 22
Deciding How Big a Resource Pool to Provide ... 22
BLDVRP: Building a Resource Pool ... 23
OPEN: Connecting a Data Set to a Resource Pool ... 25
DLVRP: Deleting a Resource Pool ... 26
Managing I/O Buffers ... 27
Deferring Write Requests ... 27
Relating Deferred Requests by Transaction ID ... 28
WRTBFR: Writing Buffers Whose Writing Has Been Deferred ... 28
Handling Exits to Physical-Error-Analysis Routines ... 30
Using the JRNAD Exit with Shared Resources ... 31
Preventing Deadlock in Exclusive Control ... 32
Using Control-Interval Access with Shared Resources ... 33
Summary of Restrictions for Shared Resources ... 37

Processing the Index of a Key-Sequenced Data Set ... 39
The Format of an Index Record ... 39
The Header Portion ... 40
The Free-Control-Interval-Entry Portion ... 40
The Index-Entry Portion ... 42
How to Gain Access to a Key-Sequenced Data Set's Index ... 46
Opening the Cluster ... 46
Opening the Index Component of the Cluster ... 46

Processing a VSAM Catalog 47
Gaining Access to a Catalog as a Catalog 47
Using the SHOWCAT Macro 47
Gaining Access to a Catalog as a Data Set 51
The Format of a VSAM Catalog 51
Opening a Catalog 52
Gaining Access to a Catalog 53

Building Parameter Lists for GENCB, MODCB, SHOWCB, and TESTCB 55
The Format of the Parameter Lists 55
Building Header and Element Entries 57
Passing Control Directly to VSAM 59
Modifying and Displaying the RECLEN Field of an RPL Directly 60
Error Return Codes 61

Appendix: Operand Notation for the Macros 63
List and Execute Forms 63
The BLDVRP Macro 63
The DLVRP Macro 64
The SHOWCAT Macro 64
Expressions That Can Be Used for Operands 66

Glossary 69

Index 71

FIGURES

Figure 1. General format of a control interval 13
Figure 2. Format of a control interval with unspanned records 15
Figure 3. Format of control intervals with spanned records 16
Figure 4. General format of an index record 40
Figure 5. Format of the header of an index record 41
Figure 6. Format of the index-entry portion of an index record 42
Figure 7. Format of an index entry 43
Figure 8. Example of key compression 44
Figure 9. Interrelationship among catalog entries 48
Figure 10. Format of header and element entries for GENCB, MODCB, SHOWCB, and TESTCB parameter lists 56

SUMMARY OF AMENDMENTS

OS/VS1 Release 5 and OS/VS2 Release 3.7

New functions have been added to VSAM as part of VS1 Release 5, and as part of VS2 Release 3.7. The changes to VSAM that are described in this publication include:

GENCB, MODCB, SHOWCB, and TESTCB

The user's program can pass control directly to VSAM, without issuing a macro, to generate, modify, display, or test an ACB, and RPL, or EXLST control block.

OS/VS1 Release 4 and OS/VS2 Independent Component

New functions have been added to VSAM as part of VS1 Release 4, and as an independent component of VS2 Release 3. For VS2 Release 3 users, the description of these new functions is for planning purposes only until they add the independent VSAM component to their system.

The changes to VSAM that are described in this publication include:

Control-Interval Access

The format of control information in a control interval has changed for relative record data sets and spanned records.

Control-interval access has a new option for improved performance. Improved control-interval access is used by way of an operand in the ACB macro.

Sharing Resources

VSAM provides new macros for sharing I/O buffers, I/O-related control blocks, and channel programs among VSAM data sets and for managing the I/O buffers. In VS1, these resources are shared in local resource pools in each partition or address space; in VS2, they may be shared in local resource pools or in a global resource pool that serves all address spaces.

Index Processing

The index of a key-sequenced data set may be processed, not only by opening it separately from the data and using the GET, PUT, and other request macros, as before, but also by opening the cluster to which it belongs and using the GETIX and PUTIX macros.

Catalog Processing

The SHOWCAT macro has been added to facilitate processing of a VSAM catalog as a catalog. It enables a program to retrieve information about a data set without opening the data set.

As in the initial release of VSAM, a catalog may be opened and processed as a key-sequenced data set. This edition discusses how to do this.

OS/VS2 Release 3

The only change that applies to VS2 Release 3 is the new discussion of opening and processing a VSAM catalog as a key-sequenced data set.

PROCESSING CONTROL INTERVALS

Control-interval access gives you access to the contents of a control interval; keyed and addressed access give you access to individual data records. The *OS/VS Virtual Storage Access Method (VSAM) Programmer's Guide,* describes keyed and addressed access and mentions control-interval access.

Control-interval access is provided for programmers of utilities and system control programs who may need more flexibility than access to data-records allows them. With control-interval access, you have the option of letting VSAM manage I/O buffers or managing them yourself. With keyed and addressed access, VSAM always manages I/O buffers. If you manage I/O buffers yourself, you have the further option of using improved control-interval access for faster processing than with normal control-interval access.

This chapter gives the format of a control interval and explains how to use control-interval access and manage your own I/O buffers.

The Format of a Control Interval

Figure 1 shows the relative positions of data, unused space, and control information in a control interval.

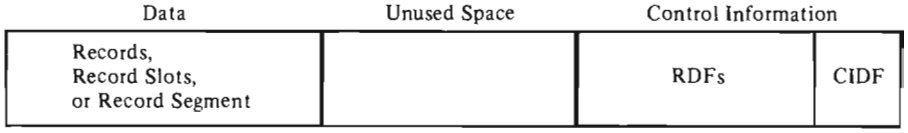


Figure 1. General format of a control interval

Control information consists of a *CIDF (control-interval definition field)* and, for a control interval containing at least one record, record slot, or record segment, one or more *RDFs (record definition fields).* The CIDF and RDFs are ordered from *right to left.*

CIDF—Control-Interval Definition Field

The CIDF is a 4-byte field that contains two 2-byte binary numbers:

Offset	Length	Description
0(0)	2	The displacement from the beginning of the control interval to the beginning of the unused space, or, if there is no unused space, to the beginning of the control information. This number is equal to the length of the data (records, record slots, or record segment). In a control interval without data, the number is 0.
2(2)	2	The length of the unused space. This number is equal to the length of the control interval, minus the length of the control information, minus the first 2-byte number. In a control interval without data (records, record slots, or record segment), the number is the length of the control interval, minus 4 (the length of the CIDF—there are no RDFs). In a control interval without unused space, the number is 0.

In an entry-sequenced data set, when there are unused control intervals beyond the last one that contains data, the first of the unused control intervals contains a CIDF filled with 0s. In a key-sequenced or relative record

data set or a key-range portion of a key-sequenced data set, the first control interval in the first unused control area (if any) contains a CIDF filled with 0s. A control interval with such a CIDF contains no data or unused space.

RDF—Record Definition Field

The RBAs of records or relative record numbers of slots in a control interval ascend from *left to right.* RDFs from *right to left* describe these records or slots or a segment of a spanned record. RDFs describe records one way for key-sequenced and entry-sequenced data sets and another way for relative record data sets.

In a *key-sequenced* or *entry-sequenced data set,* records may vary in length and may span control intervals. An unspanned record with no other records of the same length next to it is described by a single RDF that gives the length of the record. Two or more unspanned records of the same length together are described by a pair of RDFs: the RDF on the right gives the length of each record, and the RDF on the left gives their number. Each segment of a spanned record (one segment per control interval) is described by a pair of RDFs: the RDF on the right gives the length of the segment, and the RDF on the left gives its update number. (The *update number* in each segment is incremented by one each time a spanned record is updated. A difference among update numbers indicates a possible error in the record.)

In a *relative record data set,* records don't vary in length or span control intervals. Each record slot is described by a single RDF that gives its length and indicates whether it contains a record.

An RDF is a 3-byte field that contains a 1-byte control field and a 2-byte binary number:

Offset	Length and Bit Pattern	Description
0(0)	1	Control field:
	x... ..xx	Reserved.
	.x..	Indicates whether there is (1) or isn't (0) a paired RDF to the left of this RDF.
	..xx	Indicates whether the record spans control intervals: 00 No. 01 Yes—this is the first segment. 10 Yes—this is the last segment. 11 Yes—this is an intermediate segment.
	.... x...	Indicates what the 2-byte binary number gives: 0 The length of the record, segment, or slot described by this RDF. 1 The number of consecutive unspanned records of the same length, or the update number of the segment of a spanned record.
	.... .x..	For a relative record data set, indicates whether the slot described by this RDF does (0) or doesn't (1) contain a record.
1(1)	2	Binary number: When bit 4 is 0, gives the length of the record, segment, or slot described by this RDF. When bit 4 is 1 and bits 2 and 3 are 0, gives the number of consecutive records of the same length. When bit 4 is 1 and bits 2 and 3 aren't 0, gives the update number of the segment described by this RDF.

In a key-sequenced or entry-sequenced data set with *unspanned records,* the possible hexadecimal values in the control field of an RDF are:

Left RDF	Right RDF	Description
	X'00'	This RDF gives the length of a single unspanned record.
X'08'	X'40'	The right RDF gives the length of each of two or more consecutive unspanned records of the same length. The left RDF gives the number of consecutive unspanned records of the same length.

Figure 2 shows the contents of the CIDF and RDFs of a 512-byte control interval containing variable-length unspanned records.

The CIDF and the six RDFs make up 22 bytes of control information. The first 2-byte field in the CIDF gives the total length of the eight records—12x, which is the displacement from the beginning of the control interval to the unused space. The second 2-byte field gives the length of the unused space, which is the length of the control interval minus the total length of the records and the control information—512 minus 12x minus 22, or 490-12x. RDF1 and RDF2 describe the first four records; RDF3 describes the fifth record; RDF4 and RDF5 describe the sixth and seventh records; RDF6 describes the eighth record.

A control interval that contains the record segment of a *spanned record* contains no other data—it always has two RDFs. The possible hexadecimal values in their control fields are:

Left RDF	Right RDF	Description
X'18'	X'50'	The right RDF gives the length of the *first segment* of a spanned record. The left RDF gives the update number of the segment.
X'28'	X'60'	The right RDF gives the length of the *last segment* of a spanned record. The left RDF gives the update number of the segment.
X'38'	X'70'	The right RDF gives the length of an *intermediate segment* of a spanned record. The left RDF gives the update number of the segment.

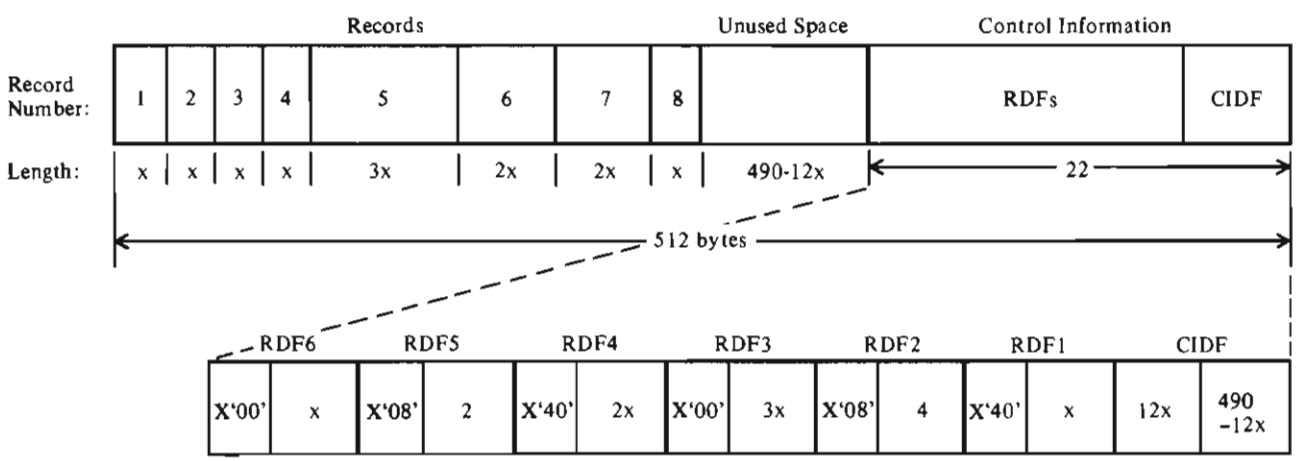


Figure 2. Format of a control interval with unspanned records

Figure 3 shows the contents of the CIDF and RDFs of three 512-byte control intervals that contain the segments of a spanned record. The number *n* in RDF2 is the update number.

Only the control interval that contains the last segment of a spanned record can have unused space. Each of the other segments uses all but the last 10 bytes of a control interval.

In a key-sequenced data set, the control intervals might not be contiguous or in the same order as the segments (that is, for example, the RBA of the second segment can be lower than the RBA of the first segment).

All of the segments of a spanned record must be in the same control area. When not enough control intervals in a control area are available for a spanned record, it is stored in a new control area.

Relative Record Data Sets

In a relative record data set, the possible hexadecimal values in the control field of an RDF are:

X'04' This RDF gives the length of an empty slot.

X'00' This RDF gives the length of a slot that contains a record.

Every control interval in a relative record data set contains the same number of slots and the same number of RDFs—one for each slot. The first slot is described by the rightmost RDF; the second slot is described by the next RDF to the left, and so on.

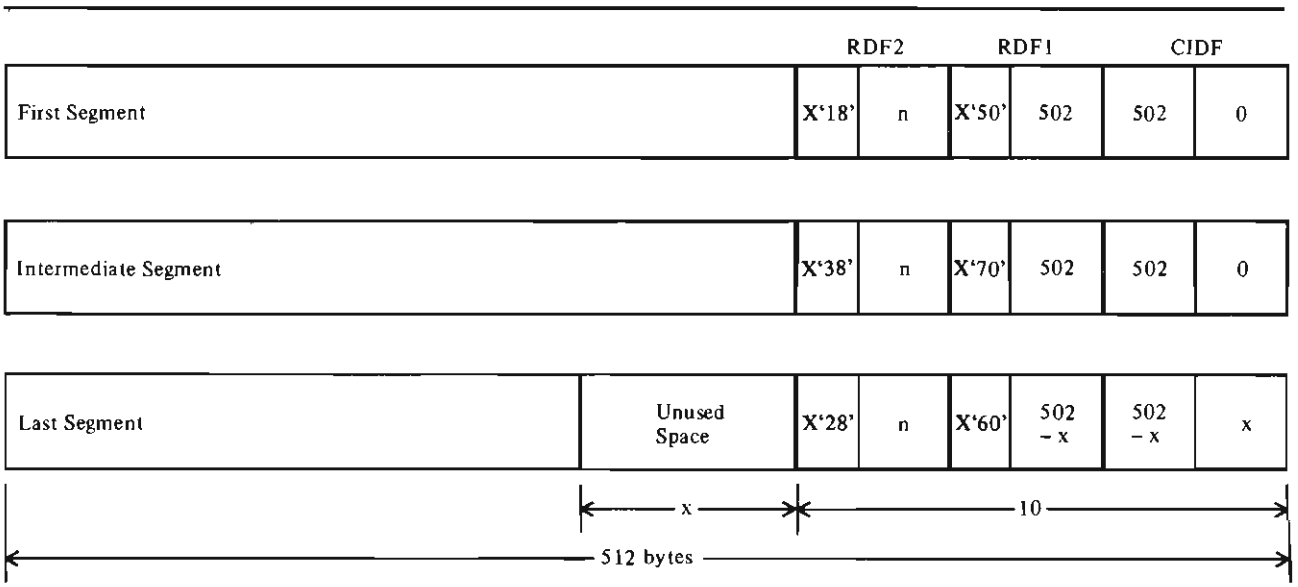


Figure 3. Format of control intervals with spanned records

How to Gain Access to a Control Interval

Control-interval access is specified entirely by the ACB (or GENCB) MACRF operand and the RPL (or GENCB) OPTCD operand. To prepare for opening a data set for control-interval access with VSAM managing I/O buffers, specify:

ACB MACRF=(CNV,...),...

NUB (no user buffering) and NCI (normal control-interval access) may be specified or taken as defaults. (This section does not consider the MACRF operands UBF (user buffering) and ICI (improved control-interval access); their use is described in the next section, "Managing Your Own I/O Buffers.")

With NUB and NCI, you may specify in the MACRF operand that the data set is to be opened for keyed and addressed access as well as for control-interval access. For example, MACRF=(CNV, KEY, SKP, DIR, SEQ, NUB, NCI, OUT) is valid.

You define a particular request for control-interval access by coding:

RPL OPTCD=(CNV,...),...

In general, control-interval access with no user buffering has the same freedoms and limitations as keyed and addressed access have. Control-interval access may be synchronous or asynchronous, may have the contents of a control interval moved to your work area (OPTCD=MVE) or left in VSAM's I/O buffer (OPTCD=LOC), and may be defined by a chain of request parameter lists (except with OPTCD=LOC specified).

Both direct and sequential access may be used with control-interval access, but skip sequential access may not. That is, you may specify OPTCD=(CNV,DIR) or (CNV,SEQ), but not OPTCD=(CNV,SKP).

With sequential access, VSAM takes an EODAD exit when you try to retrieve the control interval whose CIDF is filled with 0s or, if there is no such control interval, when you try to retrieve a control interval beyond the last one. However, VSAM does not prevent you from using a direct GET or a POINT and a sequential GET to retrieve the control interval whose CIDF is filled with 0s or any of the following control intervals.

The RPL (or GENCB) operands AREA and AREALEN have the same use for control-interval access in relation to OPTCD=MVE or LOC as they do for keyed and addressed access: with OPTCD=MVE, AREA gives the address of the area into which VSAM moves the contents of a control interval; with OPTCD=LOC, AREA gives the address of the area into which VSAM puts the address of the I/O buffer containing the contents of the control interval.

The search argument for a direct request with control-interval access is the RBA of the control interval whose contents are desired.

You can define a password specially for control-interval access. Its authority is second only to the master password; it authorizes control-interval access as well as update and read-only access. Since control-interval access gives you access to control information, you should consider protecting your data sets with control-interval passwords.

All of the request macros except ERASE can be used for control-interval access: GET, PUT, POINT, CHECK, and ENDREQ. To update the contents of a control interval, you must (with no user buffering) previously have retrieved them for update. You cannot alter the contents of a control interval with OPTCD=LOC specified.

With keyed access and addressed access, VSAM maintains the control information in a control interval. With control-interval access, you are responsible for that information.

You may load an entry-sequenced data set with control-interval access. If you open an empty entry-sequenced data set, VSAM allows you to use only sequential storage. That is, you may issue only PUTs, with OPTCD=(CNV,SEQ,NUP). PUT with OPTCD=NUP stores information in the next available control interval (at the end of the data set).

You may not load or extend a key-sequenced data set with control-interval access, because VSAM could not build index records for the data set. VSAM also prohibits you from extending a relative record data set by way of control-interval access.

To update the contents of a control interval, you can either:

- Retrieve them with OPTCD=UPD and store them back, or,
- Without retrieving them, store new contents in the control interval with OPTCD=UPD

PUT with OPTCD=UPD stores information in the control interval specified. If a GET for update precedes the PUT, specification is automatic. If no GET (or a GET with OPTCD=NUP) precedes the PUT, you have to specify the RBA of the control interval in the ARG operand.

Managing Your Own I/O Buffers

With ACB MACRF=(CNV,UBF) specified (control-interval access with user buffering), the work area specified by the RPL (or GENCB) AREA operand is, in effect, the I/O buffer—VSAM transmits the contents of a control interval directly between the work area and direct-access storage.

If you specify user buffering, you can not specify KEY or ADR in the MACRF operand—you can specify only CNV. That is, you cannot intermix keyed and addressed requests with requests for control-interval access.

OPTCD=LOC is inconsistent with user buffering and is not allowed.

Improved Control-Interval Access

With user buffering you may, if you follow the restrictions below, specify improved control-interval access:

ACB MACRF=(CNV,UBF,ICI,...),...

Its CPU (central processing unit) processing is faster than that of normal control-interval access. In OS/VS2 a processing program can achieve the best performance with improved control-interval access by combining it with SRB dispatching. (SRB dispatching is described in *OS/VS2 System Programming Library: Supervisor.)*

To open an object for improved control-interval access, the named object must:

- Be an entry-sequenced or relative record cluster; the data component of a key-sequenced, entry-sequenced, or relative record cluster; or the index component of a key-sequenced cluster (index records must not be replicated).
- Have control intervals the same size as physical records. When you use Access Method Services DEFINE to define the object, you can specify control-interval size equal to a physical-record size used for the device on which the object is stored. VSAM uses physical-record sizes of 512, 1024, 2048, and 4096 bytes. (4096 is not used, however, for the IBM 2314 Direct Access Storage Facility.)
- Not be empty.

If these restrictions aren't met, the object can't be opened.

To process a data set with improved control-interval access, a request must be:

- Defined by a single RPL (no chained request parameter lists). VSAM ignores the NXTRPL operand.
- A direct GET, GET for update, or PUT for update (no POINT, no processing empty data sets—a relative record data set with slots formatted is considered not to be empty, even if no slot contains a record).
- Synchronous (no CHECK, no ENDREQ).

With improved control-interval access, VSAM assumes (without checking) that an RPL whose ACB has MACFR=ICI has OPTCD=(CNV, DIR, SYN) that a PUT is for update (OPTCD=UPD), and that your buffer length (specified in RPL AREALEN=*number*) is correct. Since VSAM does not check these operands, you should debug your program with ACB MACRF=NCI, then change to ICI.

With improved control-interval access, VSAM doesn't take JRNAD exits and doesn't keep statistics (which are normally available by way of SHOWCB from the access-method control block).

With improved control-interval access, you may specify that control blocks are to be fixed in real storage (ACB MACRF=(CFX,...)). If you so specify, your I/O buffers must also be fixed in real storage. Having your control blocks fixed in real storage, but not your I/O buffers, may cause physical errors or unpredictable results. If you specify MACRF=CFX without ICI, VSAM ignores CFX. NFX is the default—it indicates that buffers are not fixed in real storage, except for an I/O operation. A program must be authorized to fix pages in real storage—either in supervisor state with protection key 0 or 7, or link-edited with authorization. (The authorized program facility is described in *OS/VS2 System Programming Library: Supervisor.)* An unauthorized request is ignored.

CBIC (Control Blocks in Common) Option (MVS Only)

The CBIC option places the VSAM control blocks associated with a VSAM data set into the common service area (CSA) of MVS. When you open the VSAM data set with the CBIC option, you must be in supervisor state with a protection key from 0 to 7. If this is not the case, the open will fail. The control block structure and VSAM I/O support are essentially the same with CBIC as without except for the location of the control block structure. The user-related control blocks are generated in the protection key of the user (from 0 to 7). The system-related control blocks are generated in key 0. The VSAM control block structure generated with the CBIC option retains normal control block structure connections to the region which opened the VSAM data set (such as the DEB chained to the region's TCB, etc.).

The CBIC option is invoked when a VSAM data set is opened. To invoke this option, you must set the CBIC in the ACB to one prior to OPEN. The CBIC bit is located at offset X'33' (ACBINFL2) in the ACB, bit 2 (ACBCBIC).

The CBIC option may be used only with the ICI option. The following restrictions apply when using the CBIC option:

- CBIC cannot be used with LSR or GSR specified.
- The catalog, user, or system CRA, VVIC (Virtual Volume Inventory Control), swap, or system data sets cannot be used with the CBIC option.
- A VSAM checkpoint is not allowed if CBIC data sets are being used in a region.

If another region accesses the data set's control block structure in the CSA through VSAM record management, the following conditions should be observed:

- An open must only be issued against the data set from the first region.
- The ACB of the user who opened the data set with the CBIC option must be used.
- CLOSE and CLOSE (TYPE=T) cannot be issued for the data set (only the user who opened the data set with the CBIC option can close the data set).
- A region must have the same storage protect key as the user who opened the data set with the CBIC option.
- User exit routines should be accessible from all regions accessing the data set with the CBIC option.

Error Return Codes

Some of the error return codes described in the *VSAM Programmer's Guide* have other meanings for control-interval access.

Return Codes from OPEN

One return code from OPEN in the ERROR field in the ACB has additional meanings for control-interval access:

ERROR Code	Meaning
160(A0)	The operands specified in the ACB (or GENCB) macro are inconsistent with each other or with information in the catalog. Inconsistencies include: • MACRF=ICI or UBF is specified along with LSR or GSR. • MACRF=ICI is specified, but control-interval size for the component doesn't equal physical-record size or the object to be opened is a key-sequenced cluster or an index component with record replication.
224(E0)	MACRF=ICI is specified, but the data set is empty.

Logical-Error Return Codes

Some logical-error return codes have additional meanings for control-interval access, and some apply only to control-interval access:

FDBK Code	Meaning
76(4C)	A key-sequenced or relative record data set cannot be extended by way of control-interval access.
80(50)	ERASE is invalid for control-interval access.
84(54)	OPTCD=LOC is inconsistent with MACRF=UBF.
104(68)	The RPL options are either invalid or inconsistent: • OPTCD=CNV is specified along with BWD. • With MACRF=ICI, OPTCD=(CNV, DIR,SYN) must be specified. • With MACRF=ICI, PUT with OPTCD=NUP is invalid. • With MACRF=ICI, POINT, CHECK and ENDREQ are invalid.
200(C8)	Control-interval access is not allowed for processing by way of a path.

Physical-Error Return Codes

For physical errors, the RBA field in the request parameter list gives the relative byte address of the control interval in relation to which the physical error occurred.

SHARING RESOURCES AMONG DATA SETS

VSAM has a set of macros that enable you to share I/O buffers, I/O-related control blocks, and channel programs among many VSAM data sets and to manage I/O buffers.

Sharing these resources is not the same as sharing a data set itself (that is, sharing among different tasks that independently open it). *Data-set sharing* can be done with or without sharing I/O buffers, I/O-related control blocks, and channel programs. For a discussion of data-set sharing, see the *OS/VS Virtual Storage Access Method (VSAM) Programmer's Guide.*

In VSAM, an I/O buffer is a virtual-storage area into which the contents of a control interval are read and from which data is written into a control interval. The I/O-related control blocks that are shared are: for VS1—PLH and IOB; for VS2—PLH, IOMB, IOSB, IQE, IRB, and SRB.

Sharing these resources optimizes their use to reduce the requirement for virtual storage and therefore to reduce paging of virtual storage.

Managing I/O buffers includes:

- Deferring writes for direct PUT requests, which reduces the number of I/O operations
- Writing buffers that have been modified by related requests
- Locating buffers that contain the contents of specified control intervals
- Marking a buffer to be written without issuing a PUT

Managing I/O buffers should enable you to speed up direct processing of VSAM data sets whose activity is unpredictable. You probably won't be able to speed up sequential processing or processing of a data set whose activity is consistently heavy.

When you share resources for sequential access, positioning at the beginning of a data set has to be explicit: with the POINT macro or the direct GET macro with RPL OPTCD=NSP. With shared resources, VSAM doesn't automatically position itself at the beginning of a data set opened for sequential access, because placeholders belong to the resource pool, not to individual data sets. You may not use a resource pool to load records into an empty data set. (Sharing wouldn't improve sequential processing, but it would take up buffers and could hinder other processing.)

The macros you use to share resources and manage I/O buffers are:

- BLDVRP (build VSAM resource pool)
- DLVRP (delete VSAM resource pool)
- WRTBFR (write buffer)
- SCHBFR (search buffer)
- MRKBFR (mark buffer)

Certain operands in the ACB and RPL macros are used with these macros. The use of these macros and operands and of the SHOWCAT, SHOWCB, and TESTCB macros is described in this chapter. The appendix explains operand notations for coding macros.

Providing a Resource Pool

To share resources, you use the BLDVRP macro to build a resource pool. You code a MACRF operand in the ACB and use OPEN to connect your data sets to the resource pool. After you have closed all of the data sets, you use the DLVRP macro to delete the resource pool.

You may share resources locally or globally:

- LSR (local shared resources). Each partition or address space may have one resource pool independently of other partitions or address spaces.
- GSR (global shared resources), VS2 only. All address spaces in the system share one resource pool. With GSR, an access-method control block and all related request parameter lists, exit lists, data areas, and extent control blocks must be in the common area of virtual storage with protection key 7. To get storage in the common area with protection key 7, issue the GETMAIN macro, while in that key, for storage in subpool 231. Generate ACBs, RPLs, and EXLSTs with the GENCB macro—code the WAREA and LENGTH operands. The program that issues macros related to the global resource pool must be in supervisor state with key 0 or 7. (The macros are: BLDVRP, CHECK, CLOSE, DLVRP, ENDREQ, ERASE, GENCB, GET, GETIX, MODCB, MRKBFR, OPEN, POINT, PUT, PUTIX, SCHBFR, SHOWCB, TESTCB, and WRTBFR. The SHOWCAT macro is not related to a resource pool, because it is issued independently of an open data set.)

In VS2, you may have both a global resource pool and local resource pools. Tasks in an address space that has a local resource pool may use either the global resource pool, under the restrictions described above, or the local resource pool.

Deciding How Big a Resource Pool to Provide

You have to provide a resource pool before any clusters or alternate indexes are opened to use it. Specifying the BUFFERS, KEYLEN, and STRNO operands of the BLDVRP macro requires knowledge of the size of the control intervals, data records (if spanned), and key fields in the components that will use the resource pool and knowledge about the way the components are processed. (See "Format of the BLDVRP Macro" below.)

Displaying Information about an Unopened Data Set

The SHOWCAT macro enables you to get information about a component before its cluster or alternate index is opened. The program that is to issue BLDVRP can issue SHOWCAT on all of the components to find out the sizes of control intervals, records, and keys. This information enables the program to calculate values for the BUFFERS and KEYLEN operands of BLDVRP.

A program need not be in supervisor state with key 0 or 7 to issue SHOWCAT, even though it must be in supervisor state to issue BLDVRP TYPE=GSR (VS2 only).

The SHOWCAT macro is described in "Using the SHOWCAT Macro" in the chapter "Processing a VSAM Catalog."

Displaying Statistics about a Buffer Pool

You can get statistics about the usage of buffer pools to determine how you could improve a previous definition of a resource pool and the mix of data sets that use it. The SHOWCB macro enables you to get statistics about a buffer pool. The statistics are available from an ACB that describes an open data set that is using the buffer pool. They reflect the usage of the buffer pool from the time it was built to the time SHOWCB is issued. All of the statistics except one are for a single buffer pool. To get statistics for the whole resource pool, issue SHOWCB for each buffer pool in it.

The statistics cannot be used to redefine the resource pool while it is in use. You have to make adjustments the next time you build it.

The use of SHOWCB to display an ACB is described in the *VSAM Programmer's Guide*. If the ACB has MACRF=GSR (VS2 only), the program that issues SHOWCB must be in supervisor state with protection key 0 or 7. For buffer-pool statistics, the following keywords are specified in the FIELDS operand:

SHOWCB FIELDS=([BFRFND][,BUFRDS][,NUIW][,STRMAX][,UIW],...),...

These fields may be displayed only after the data set described by the ACB is opened. Each field requires one fullword in the display work area:

Field	Description
BFRFND	The number of requests for retrieval that could be satisfied without an I/O operation (the data was found in a buffer).
BUFRDS	The number of reads to bring data into a buffer.
NUIW	The number of non-user-initiated writes (writes that VSAM was forced to do because no buffers were available for reading the contents of a control interval).
STRMAX	The maximum number of placeholders currently active for the resource pool (for all of the buffer pools in it).
UIW	The number of user-initiated writes (PUTs not deferred or WRTBFRs—see "Deferring Write Requests" below).

Return Codes from SHOWCB. The return codes from VSAM for SHOWCB with these keywords are the same as for other keywords and are given in the *VSAM Programmer's Guide*. With MACRF=GSR specified in the ACB, a program check can occur if SHOWCB is issued by a program that is not in supervisor state with protection key 0 or 7.

BLDVRP: Building a Resource Pool

To share resources locally, a task in the partition or address space issues the BLDVRP (build VSAM resource pool) macro, TYPE=LSR. To share resources globally, a system task issues BLDVRP, TYPE=GSR. Issuing BLDVRP causes VSAM to share the I/O buffers, I/O-related control blocks, and channel programs of data sets whose ACBs indicate the corresponding option for shared resources. (See "OPEN:Connecting a Data Set to a Resource Pool" below.) Control blocks and channel programs are shared automatically. But you may control the sharing of buffers.

When you issue BLDVRP, you specify the number of buffer pools in the resource pool and the size and number of buffers in each buffer pool. A data set uses the buffer pool whose buffers are the appropriate size—either the

exact size needed, if there are any, or the next-larger size available. The data set uses only the one buffer pool.

Format of the BLDVRP Macro

The BLDVRP macro has a standard form and list and execute forms. The standard form builds a parameter list and passes control to VSAM to build the resource pool. The list and execute forms are described in the appendix.

The format of the BLDVRP macro is:

BLDVRP	**BUFFERS=**(*size* (*number*),*size* (*number*),...) [**FIX=**{**BFR** | **IOB** | **(BFR,IOB)**}] [**KEYLEN=** *length*] **STRNO=** *number* [**TYPE=**{**LSR** | **GSR**}]

BUFFERS=(*size* (*number*), *size* (*number*),...)
specifies the size and number of buffers in each buffer pool in the resource pool. The number of buffer pools in the resource pool is implied by the number of *size(number)* pairs you specify.

When you process a key-sequenced data set, the index component, as well as the data component, shares the buffers of a buffer pool. When you use an alternate index to process a base cluster, the components of the alternate index and the base cluster share buffers. The components of alternate indexes in an upgrade set share buffers. Buffers of the appropriate size and number must be provided for all of these components, each of which uses the buffer pool whose buffers are exactly the right size or the next-larger size.

size
is 512, 1024, 2048, 4096, or so on in increments of 4096 to a maximum of 32K bytes.

number
is at least 3.

FIX= { **BFR** | **IOB** | **(BFR,IOB)** }
specifies that I/O buffers (BFR), or I/O-related control blocks (IOB), or both, are to be fixed in real storage. With GSR, IOB includes channel programs. If the program that issues BLDVRP with FIX specified is not authorized to fix areas in real storage, FIX is ignored. A program is authorized if it is in supervisor state with protection key 0 or 7, or has been link-edited with authorization (the authorized program facility is described in *OS/VS2 System Programming Library: Supervisor*).

KEYLEN= *length*
specifies the maximum key length of the data sets that are to share the resource pool. The default is 255. The keys whose lengths must be provided for are the prime key of each key-sequenced data set and the alternate key of each alternate index that is used for processing or is upgraded. If none of the data sets is keyed, specify 0.

STRNO= *number*
specifies the total number of placeholders required for all of the data sets that are to share the resource pool. 1 is minimum; 255 is maximum.

The number should equal the potential number of requests that may be issued concurrently for all of the data sets that will share the resource pool.

If a request fails because the number of placeholders is insufficient (logical return code 64—X'40'), you may retry the request; it will be assigned a placeholder if one has been released.

TYPE= { LSR | GSR }
specifies whether a local (LSR) or a global (GSR) resource pool is to be built. Only one BLDVRP TYPE=LSR may be issued per partition or address space; only one BLDVRP TYPE=GSR (VS2 only) may be issued for the system. The program that issues BLDVRP TYPE=GSR must be in supervisor state with key 0 or 7.

Return Codes from BLDVRP

VSAM returns a code in register 15 that indicates whether the BLDVRP request was successful:

Reg. 15	Meaning
0(0)	VSAM completed the request.
4(4)	A resource pool already exists in the partition or address space (LSR) or in the system (GSR).
8(8)	There isn't enough virtual-storage space to satisfy the request. GETMAIN or ESTAE (VS2 only) failed.
12(C)	Buffers can't be fixed in real storage. PAGEFIX failed.
16(10)	VS1: TYPE=GSR is specified. VS2: TYPE=GSR is specified but the program that issued BLDVRP isn't in supervisor state with key 0 or 7.
20(14)	STRNO is less than 1 or greater than 255.
24(18)	BUFFERS is specified incorrectly. A *size* or *number* is invalid.

OPEN: Connecting a Data Set to a Resource Pool

You cause a data set to use a resource pool that has been built by BLDVRP by specifying LSR or GSR in the MACRF operand of the data set's ACB before you open the data set.

ACB MACRF=({ NSR | LSR | GSR } ,...),...

NSR, the default, indicates the data set will not use shared resources. LSR indicates it will use the local resource pool. GSR indicates it will use the global resource pool (VS2 only).

When an ACB indicates LSR or GSR, VSAM ignores its BSTRNO, BUFNI, BUFND, BUFSP, and STRNO operands—it has used the BUFFERS and STRNO values that were specified in BLDVRP.

For a data set described by an ACB with MACRF=GSR (VS2 only), the ACB and all related RPLs, EXLSTs, ECBs, and data areas must be in the common area of virtual storage with protection key 7, and all VSAM requests related to the global resource pool (including OPEN) must be issued in supervisor state with protection key 0 or 7. Open assumes that control blocks and data areas are in the common area with protection key 7—if they aren't, a program check can occur during subsequent processing.

Return Codes from OPEN

The standard return codes from OPEN are described in the *VSAM Programmer's Guide*. In addition to those, you may get other return codes in the ACB ERROR field when you try to open a data set with shared resources. An 8 in register 15 indicates that at least one data set wasn't opened successfully, and the ERROR field gives the reason:

ERROR Code	Meaning
160(A0)	The operands specified in the ACB or GENCB macro are inconsistent with each other or with the information in the catalog record. With shared resources, this code can mean: • MACRF options are inconsistent: LSR or GSR is specified with ICI or UBF (see "Using Control-Interval Access with Shared Resources" below), or DFR is specified without LSR or GSR (see "Deferring Write Requests" below) • MACRF DFR is specified for a data set that was defined with SHAREOPTION 4 (see "Deferring Write Requests" below)
204(CC)	VS2 only: MACRF=GSR is specified but the program that issued OPEN isn't in supervisor state with protection key 0 or 7,
208(D0)	VS1 only: MACRF=GSR is specified.
212(D4)	MACFR=LSR or GSR is specified but the data set being opened is empty (which implies that the data set will be loaded).
216(D8)	MACRF=LSR or GSR is specified but the key length of the data set being opened is greater than the maximum key length specified in BLDVRP for the resource pool.
220(DC)	MACRF=LSR or GSR is specified but the control-interval size of the data set being opened is greater than the largest buffer size specified in BLDVRP for the resource pool.

DLVRP: Deleting a Resource Pool

After all data sets using a resource pool have been closed, you delete the resource pool by issuing the DLVRP (delete VSAM resource pool) macro. Failure to delete a local resource pool causes virtual storage to be lost until the end of the task. OS/VS2 protects itself against this loss with a global resource pool: if the address space that issued BLDVRP terminates without having issued DLVRP, the system deletes the global resource pool when its use count is 0.

Format of the DLVRP Macro

The DLVRP macro has a standard form and an execute form. The standard form builds a parameter list and passes control to VSAM to delete the resource pool. The execute form is described in the appendix.

The format of the DLVRP macro is:

DLVRP	**TYPE={LSR | GSR}**

TYPE= { LSR | GSR }
specifies the type of resource pool to be deleted: local (LSR) or global (GSR). The local resource pool is the one in the partition or address space in which DLVRP is issued. The program that issues DLVRP TYPE=GSR | (VS2 only) must be in supervisor state with protection key 0 or 7.

Return Codes from DLVRP

VSAM returns a code in register 15 that indicates whether the DLVRP request was successful:

Reg. 15	Meaning
0(0)	VSAM completed the request.
4(4)	There is no resource pool to delete.
8(8)	There isn't enough virtual-storage space to satisfy the request. GETMAIN or ESTAE (VS2 only) failed.
12(C)	There is at least one open data set using the resource pool.
16(10)	VS1: TYPE=GSR is specified. VS2: TYPE=GSR is specified, but the program that issued DLVRP isn't in supervisor state with protection key 0 or 7.

Managing I/O Buffers

Managing I/O buffers includes:

- Deferring writes for direct PUT requests
- Writing buffers that have been modified by related requests
- Locating buffers that contain the contents of specified control intervals
- Marking a buffer to be written without issuing a PUT

Deferring Write Requests

VSAM automatically defers writes for sequential PUT requests. It normally writes out the contents of a buffer immediately for direct PUT requests. With shared resources, you can cause writes for direct PUT requests to be deferred. Buffers are finally written out:

- When you issue the WRTBFR macro
- When VSAM needs a buffer to satisfy a GET request
- Or when a data set using a buffer pool is closed (temporary CLOSE is ineffective—a no-op—against a data set that is sharing buffers; nor does ENDREQ cause buffers in a resource pool to be written)

Deferring writes saves I/O operations in cases where subsequent requests can be satisfied by the data in the buffer pool. Data processing performance with VSAM will be improved if control intervals are updated more than once.

You indicate that writes are to be deferred by coding the MACRF DFR option in the ACB, along with MACRF=LSR or GSR.

ACB **MACRF=({ LSR | GSR },{DFR | NDF } ,...),...**

NDF, the default, indicates that writes are not to be deferred for direct PUTs.

The DFR option is incompatible with SHAREOPTION 4. (SHAREOPTION is a parameter of the DEFINE command of Access Method Services. It is described in the appropriate *Access Method Services*. A request to open a data set with SHAREOPTION 4 for deferred writes is rejected. (See "Return Codes from OPEN" above.)

Relating Deferred Requests by Transaction ID

You can relate action requests (GET, PUT, etc.) according to transaction by specifying the same ID in the RPLs that define the requests.

The purpose of relating the requests that belong to a transaction is to enable WRTBFR to cause all of the modified buffers used for a transaction to be written. When the WRTBFR request is complete, the transaction is physically complete.

RPL **TRANSID=** *number*,...

TRANSID= *number*
specifies a number from 0 to 31. A number from 1 to 31 relates the request(s) defined by this RPL to the requests defined by other RPLs with the same transaction ID. The number 0, which is the default, indicates that requests defined by the RPL are not associated with other requests.

You can find out what transaction ID an RPL has by issuing SHOWCB or TESTCB. Their use with an RPL is described in the *VSAM Programmer's Guide.*

SHOWCB **FIELDS=([TRANSID] ,...),...**

TRANSID requires one fullword in the display work area.

TESTCB **TRANSID=** *number*,...

If the ACB to which the RPL is related has MACRF=GSR (VS2 only), the program that issues SHOWCB or TESTCB must be in supervisor state with protection key 0 or 7.

Return Codes from SHOWCB and TESTCB. The return codes from VSAM for SHOWCB and TESTCB with the TRANSID keyword are the same as with other keywords and are given in the *VSAM Programmer's Guide.* With MACRF=GSR specified in the ACB to which the RPL is related, a program check can occur if SHOWCB or TESTCB is issued by a program that is not in supervisor state with protection key 0 or 7.

WRTBFR: Writing Buffers Whose Writing Has Been Deferred

If DFR is specified in the ACB of any data set that is using a resource pool, you can use the WRTBFR (write buffer) macro to write:

- All modified buffers for a given data set
- All modified buffers in the resource pool
- The least recently used modified buffers in each buffer pool in the resource pool
- All buffers that have been modified by requests with the same transaction ID (see "Relating Deferred Requests by Transaction ID" above)

You can specify the DFR option in an ACB without using the WRTBFR to write buffers—a buffer will be written when VSAM needs one to satisfy a GET request, or all modified buffers will be written when the last of the data sets that uses them is closed.

Besides using WRTBFR to write buffers whose writing has been deferred, you can use it to write buffers that have been marked for output with the MRKBFR macro, which is described in "MRKBFR:Marking a Buffer for Output" below.

Using WRTBFR can improve performance, if you schedule WRTBFR to overlap other processing.

VSAM notifies the processing program when there are no more unmodified buffers into which to read the contents of a control interval. (VSAM would be forced to write buffers when another GET request required an I/O operation.) VSAM sets register 15 to 0 and puts 12 (X'0C') in the feedback field of the RPL that defines the PUT request that detects the condition.

VSAM also notifies the processing program when there are no buffers available to which to assign a placeholder for a request. This is a logical error (register 15 contains 8 unless an exit is taken to a LERAD routine); the feedback field in the RPL contains 152 (X'98'). You may retry the request; it will get a buffer if one has been freed.

Format of the WRTBFR Macro

The format of the WRTBFR macro is:

WRTBFR	**RPL=***address* **TYPE={ALL | CHK | DS | LRU(***percent***) | TRN}**

RPL= *address*
specifies the address of the request parameter list that defines the WRTBFR request. An RPL need not be built especially for the WRTBFR—WRTBFR may use an inactive RPL that defines other request(s) (GET, PUT, etc.) for a data set that is using the resource pool. These RPL operands have meaning for WRTBFR:

- **ACB=** *address.*
- **ECB=** *address.*
- **OPTCD=** { **ASY** | **SYN** }.WRTBFR can be issued synchronously (SYN) or asynchronously (ASY). A CHECK or ENDREQ must be issued to synchronize an asynchronous WRTBFR request.
- **TRANSID=** *number.* Specifies a number from 0 to 31.

All other RPL operands are ignored. RPLs are assumed not to be chained.

If the ACB to which the RPL is related has MACRF=GSR (VS2 only), the program that issues WRTBFR must be in supervisor state with protection key 0 or 7.

TYPE={ALL | CHK | DS | LRU(*percent***) | TRN}**
specifies what buffers are to be written.

ALL
specifies that all modified buffers in each buffer pool in the resource pool are to be written. Closing all of the data sets that use a resource pool causes the same buffers to be written.

CHK
same as TRN (below), but if an error occurs in writing buffers, transaction IDs continue to be associated with the buffers. WRTBFR TYPE=CHK could be used by a checkpoint routine to record checkpoint information and leave buffers for which an error occurred as they were for continued processing.

DS
specifies that, for the data set defined by the ACB to which the WRTBFR's RPL is related, all modified buffers are to be written and marked empty.

LRU(*percent* **)**
specifies that some of the modified buffers in each buffer pool in the resource pool are to be written. The *percent* is the percent of buffers in each pool that are to be examined for possible writing. The least recently used buffers are examined. (If *percent* is coded in register notation, only registers 1 and 13 may not be used.)

TYPE=LRU is used for writing some modified buffers, without respect to a particular data set or transaction ID, to ensure that buffers will be available for GET requests (without having to wait for buffers to be written).

TRN
specifies that all buffers in a buffer pool are to be written that have been modified by requests with the transaction ID specified in the WRTBFR's RPL. Transaction IDs are no longer associated with these buffers.

Return Codes from WRTBFR

There are no special return codes from WRTBFR. Many of the standard logical-error codes apply. They are described in the *VSAM Programmer's Guide*. Physical-error codes are passed only to an EXCEPTION, JRNAD, or SYNAD exit routine (see "Handling Exits to Physical-Error-Analysis Routines" below). With GSR, a program check can occur if WRTBFR is issued by a program that is not in supervisor state with protection key 0 or 7.

Handling Exits to Physical-Error-Analysis Routines

With deferred writes of buffers, a processing program continues after its PUT request has been completed, even though the buffer hasn't been written. The processing program is not synchronized with a physical error that occurs when the buffer is finally written. A processing program that uses MRKBFR MARK=OUT (see "MRKBFR:Marking a Buffer for Output" below) is also not synchronized with a physical error. An EXCEPTION or a SYNAD exit routine must be supplied to analyze the error.

The *ddname* field of the physical-error message identifies the data set that was using the buffer, but because the buffer might have been released, its contents might be unavailable. You can provide a JRNAD exit routine to record the contents of buffers for I/O errors (see "Using the JRNAD Exit with Shared Resources" below). It can be co-ordinated with a physical-error-analysis routine to handle I/O errors for buffers whose writing has been deferred.

If a JRNAD exit routine is used to cancel I/O errors during a transaction, the physical-error-analysis routine will get only the last error return code.

Using the JRNAD Exit with Shared Resources

In addition to the reasons for having a JRNAD exit routine that are described in the *VSAM Programmer's Guide,* VSAM takes the JRNAD exit for the following reasons when the exit is associated with a data set whose ACB has MACRF=LSR or GSR:

- A data or index control-interval buffer has been modified and is about to be written.
- A physical error occurred: VSAM takes the JRNAD exit first—your routine can direct VSAM to bypass the error and continue processing or to terminate the request that occasioned the error and proceed with error processing.
- A control area is about to be split.

The *VSAM Programmer's Guide* gives the contents of the registers when VSAM exits to the JRNAD routine. Some of the fields in the parameter list that register 1 points to have special meanings for buffer modification, I/O errors, and control-area splits:

Offset	Length	Description	
0(0)	4	No special meaning. (Address of the RPL that defines the request that occasioned VSAM's exit to the routine.)	
4(4)	4	No special meaning. (Address of a 5-byte field that identifies the data set being processed.)	
8(8)	4	Variable, depends on the reason indicator at offset 20:	
		X'20'	The RBA of the beginning of the control area about to be split.
		X'24'	The address of the I/O buffer into which data was going to be read.
		X'28'	The address of the I/O buffer from which data was going to be written.
		X'2C'	The address of the I/O buffer that contains the control-interval contents that are about to be written.
12(C)	4	Variable, depends on the reason indicator at offset 20:	
		X'20'	Unpredictable.
		X'24'	Unpredictable.
		X'28'	Bits 0 through 31 correspond with transaction IDs 0 through 31. Bits set to 1 indicate that the buffer that was being written when the error occurred was modified by the corresponding transactions. You can set additional bits to 1 to communicate to VSAM to keep the contents of the buffer until the corresponding transactions have modified the buffer.
		X'2C'	The size of the control interval whose contents are about to be written.
16(10)	4	Variable, depends on the reason indicator at offset 20:	
		X'20'	The RBA of the last byte in the control area about to be split.
		X'24'	The fourth byte contains the physical-error code from the RPL FDBK field. You use this fullword to communicate with VSAM. Setting it to 0 indicates that VSAM is to

Offset	Length	Description	
			ignore the error, bypass error processing, and let the processing program continue. Leaving it non0 indicates that VSAM is to continue as usual: terminate the request that occasioned the error and proceed with error processing, including exiting to a physical-error-analysis routine.
		X'28'	Same as for X'24'.
		X'2C'	The RBA of the control interval whose contents are about to be written.
20(14)	1	Indication of the reason VSAM exited to the JRNAD routine:	
		X'20'	A control area is about to be split. During the split, VSAM exits to the JRNAD routine as each control interval that is modified is about to be written (see X'2C' below).
		X'24'	A physical input error occurred.
		X'28'	A physical output error occurred.
		X'2C'	A modified data or index control interval is about to be written.
21(15)	1	Reserved.	

Preventing Deadlock in Exclusive Control

Contention for VSAM data (the contents of a control interval) can lead to deadlocks, in which a processing program is prevented from continuing because its request for data cannot be satisfied. Two contenders, A and B, can engage in four distinct contests:

1. A wants exclusive control, but B has exclusive control. VSAM refuses A's request—A must either do without the data or retry its request.
2. A wants exclusive control, but B is only willing to share. VSAM queues A's request (without notifying A that it must wait) and gives A use of the data when B releases it.
3. A wants to share, but B has exclusive control. VSAM refuses A's request—A must either do without the data or retry its request.
4. A wants to share, and B is willing to share. VSAM gives A use of the data, along with B.

VSAM's action in a contest for data rests on the assumptions that, if a processing program has exclusive control of the data, it will (or at least might) update or delete it and that, if a processing program is updating or deleting the data, it has exclusive control of it. (The use of MRKBFR, MARK=OUT, provides an exception to the second assumption: a processing program can update the contents of a control interval without exclusive control of them—see "MRKBFR:Marking a Buffer for Output" below.)

In contests 1 and 3, B is responsible for giving up exclusive control of a control interval by way of an ENDREQ, a MRKBFR, MARK=RLS, or a request for access to a different control interval. (The RPL that defines the ENDREQ, MRKBFR, or request is the one that was used to acquire exclusive control in the first place.)

Using Control-Interval Access with Shared Resources

When you share resources, you can use two macros (SCHBFR and MRKBFR) to locate a buffer in a buffer pool and to mark it for output. You can update its contents (which are the contents of a control interval) independently of much of VSAM record management. Doing this might improve your data processing performance. It allows you to bypass some of the restrictions VSAM imposes on the way you treat VSAM data sets.

However, bypassing normal restrictions might prevent you from processing the data sets normally. For instance, key-sequenced data sets depend on the accuracy of their indexes and the RDFs and the CIDF in each control interval. You should not update key-sequenced data sets with control-interval access.

Control-interval access with SCHBFR and MRKBFR is intended primarily for processing entry-sequenced data sets. The definition of an entry-sequenced data set includes the unchangeability of RBAs and the permanence of records. With control-interval access, you can change RBAs in a control interval and delete records by modifying the RDFs and the CIDF. However, if the data set has alternate indexes, you are responsible for upgrading them. Upgrading an alternate index is described in the *VSAM Programmer's Guide.*

SCHBFR (search buffer) enables you to locate buffers that contain the contents of adjacent control intervals.

MRKBFR (mark buffer) enables you to mark a buffer for output. You can modify the contents of the buffer with or without having exclusive control of it, and, with WRTBFR, you can cause the modified buffer to be written without issuing a PUT for update.

MRKBFR also enables you to release a buffer from exclusive control or shared status if it was under exclusive control or in shared status. If the buffer was marked to be written, it is still marked to be written. The placeholder associated with the RPL that defines MRKBFR is kept for further use by the RPL, but it no longer holds a place (positioning is invalidated).

Improved control-interval access (ACB MACRF=(ICI,...) and standard user buffering (RPL OPTCD=UBF,...) are not allowed with shared resources (that is, where ACB MACRF= ({LSR | GSR},...). (See "Return Codes from OPEN" above.)

SCHBFR: Locating an RBA in a Buffer Pool

When a resource pool is built, the buffers in each buffer pool are numbered from 1 through the number of buffers in each buffer pool. At a given time, several buffers in a buffer pool may hold the contents of control intervals for a particular data set. These buffers may or may not contain RBAs of interest to your processing program. The SCHBFR macro enables you to find out. You specify in the ARG operand of the RPL that defines SCHBFR the address of an 8-byte field that contains the first and last control-interval RBAs of the range you're interested in.

The buffer pool to be searched is the one used by the data component defined by the ACB to which your RPL is related. If the ACB names a path, VSAM searches the buffer pool used by the data component of the alternate index. (If the path is defined over a base cluster alone, VSAM searches the buffer pool used by the data component of the base cluster.) VSAM begins its search at the buffer you specify and continues until it finds a buffer that

contains an RBA in the range or until the highest-numbered buffer has been searched.

You should begin a search at the first buffer in the buffer pool. If a buffer satisfies the search, VSAM returns its address (OPTCD=LOC) or its contents (OPTCD=MVE) in the work area whose address is specified in the AREA operand of the RPL and returns its number in register 0. To find the next buffer that contains an RBA in the range, issue SCHBFR again and specify the number of the next buffer after the first one that satisfied the search. You continue until VSAM indicates that it found no buffer that contains an RBA in the range or until you reach the end of the pool (see "Return Codes from SCHBFR" below).

Finding a buffer that contains a desired RBA doesn't get you exclusive control of the buffer. You may get exclusive control only by issuing GET for update. SCHBFR doesn't return the location or the contents of a buffer that is already under the exclusive control of another request.

Format of the SCHBFR Macro

The format of the SCHBFR macro is:

SCHBFR	[**BFRNO=** *number*] **RPL=** *address*

BFRNO= *number*
specifies the number of the buffer VSAM is to search first. The buffers preceding it in the buffer pool are not searched. The default is 1; that is, the first buffer is searched first. (If *number* is coded in register notation, only registers 1 and 13 may not be used.)

RPL= *address*
specifies the address of the request parameter list that defines the SCHBFR request. These RPL operands have meaning for SCHBFR:

- **ACB=** *address*.
- **AREA=** *address*. If a buffer is found, the area whose address is specified will contain its address (OPTCD=LOC) or a copy of its contents (OPTCD=MVE).
- **AREALEN=** *number*. At least 4 with OPTCD=LOC; at least control-interval size with OPTCD=MVE.
- **ARG=** *address*. ARG gives the address of an 8-byte field that contains the beginning and ending control-interval RBAs of the range to be searched on.
- **ECB=** *address*.
- **OPTCD=({ ASY | SYN } , { LOC | MVE }).**
- **TRANSID=** *number*.

All other RPL operands are ignored. RPLs are assumed not to be chained. Control-interval access is assumed.

If the ACB to which the RPL is related has MACRF=GSR (VS2 only), the program that issues SCHBFR must be in supervisor state with
| protection key 0 or 7.

Return Codes from SCHBFR

VSAM indicates that it found a buffer that satisfies SCHBFR, with 0 in register 15 and the number of the buffer in register 0. There are no other special return codes from SCHBFR. Some of the standard codes apply; the following logical-error codes, with 8 in register 15 (unless a LERAD exit routine is supplied), have a special meaning:

FDBK Code	Meaning
16(10)	No buffer that was searched contains a control-interval RBA in the range specified.
44(2C)	The work area isn't large enough for the buffer (OPTCD=MVE).
104(68)	The RPL options are either invalid or inconsistent; BFRNO may be invalid (less than 1 or greater than the number of buffers in the pool).

Other standard codes are described in the *VSAM Programmer's Guide*. With GSR, a program check can occur if SCHBFR is issued by a program that is not in supervisor state with protection key 0 or 7.

MRKBFR: Marking a Buffer for Output

You locate a buffer that contains the RBA you're interested in by issuing a SCHBFR macro, a read-only GET, or a GET for update. When you issue GET for update, you get exclusive control of the buffer. Whether you have exclusive control or not, you can mark the buffer for output by issuing the MRKBFR (mark buffer) macro with MARK=OUT and then change the buffer's contents. Without exclusive control, you should not change the control information in the CIDF or RDFs (that is, you should not change record lengths).

MRKBFR, MARK=OUT, indicates that the buffer's contents are modified. You must modify the contents of the buffer itself—not a copy. Consequently, when you issue SCHBFR or GET to locate the buffer, you must specify RPL OPTCD=LOC. (If you use OPTCD=MVE, you get a copy of the buffer but don't learn its location.) The buffer will be written when a WRTBFR is issued or when VSAM is forced to write a buffer to satisfy a GET request.

If you are sharing a buffer or have exclusive control of it, you can release it from shared status or exclusive control with MRKBFR, MARK=RLS. If the buffer was marked for output, MRKBFR with MARK=RLS doesn't nullify it—the buffer will eventually be written. Sequential positioning is lost. MRKBFR with MARK=RLS is similar to the ENDREQ macro.

Format of the MRKBFR Macro

The format of the MRKBFR macro is:

MRKBFR	**MARK={OUT | RLS}** **RPL=** *address*

MARK= { OUT | RLS }
specifies whether to mark for output or to release from exclusive control or shared status the buffer identified in the RPL. To do both, issue MRKBFR twice, once with MARK=OUT, again with MARK=RLS.

OUT
indicates that the buffer is to be marked for output. The buffer is kept under exclusive control or in shared status.

RLS
indicates that the buffer is to be released from exclusive control or shared status.

RPL= *address*
specifies the address of the request parameter list that defines the MRKBFR request. Use the RPL used by SCHBFR or GET to locate the buffer being marked or released. These RPL operands have meaning for MRKBFR:

- **ACB=** *address.*
- **ECB=** *address.*
- **TRANSID=** *number.*

All other RPL operands are ignored. RPLs are assumed not to be chained. OPTCD=LOC is assumed.

If the ACB to which the RPL is related has MACRF=GSR (VS2 only), the program that issues MRKBFR must be in supervisor state with protection key 0 or 7.

Return Codes from MRKBFR

There are no special return codes from MRKBFR. Some of the standard codes apply. They are described in the *VSAM Programmer's Guide.* With GSR, a program check can occur if MRKBFR is issued by a program that is not in supervisor state with protection key 0 or 7.

Summary of Restrictions for Shared Resources

Restrictions for using the LSR and GSR options are:

- Empty data sets cannot be processed (that is, loaded).
- Improved control-interval access cannot be used (ACB MACRF=ICI).
- Standard user buffering is not allowed (RPL OPTCD=UBF).
- Writes for data sets with SHAREOPTION 4 cannot be deferred (ACB MACRF=DFR).
- Request parameter lists for MRKBFR, SCHBFR, and WRTBFR cannot be chained (the NXTRPL operand of the RPL macro is ignored).
- For sequential access, positioning at the beginning of a data set must be explicit: with a POINT macro or a direct GET macro with RPL OPTCD=NSP.
- Temporary CLOSE and ENDREQ don't cause buffers to be written if MACRF=DFR was specified in the associated ACB.
- GSR is for VS2 only. With GSR, an ACB and all related RPLs, EXLSTs, data areas, and ECBs must be stored in the common area of virtual storage with protection key 7; all VSAM requests related to the global resource pool may be issued only by a program in supervisor state with protection key 0 or 7.
- Checkpoints cannot be taken for data sets whose resources are shared in a *global resource pool.* When a program in an address space that opened a data set whose ACB has MACRF=GSR issues the CHKPT macro, 8 is returned in register 15. If a program in another address space issues the CHKPT macro, the checkpoint is taken, but only for data sets that are not using the global resource pool.

  Checkpoint/restart can be used with data sets whose resources are shared in a *local resource pool,* but the restart program doesn't reposition for processing at the point where the checkpoint occurred—processing is restarted at a data set's highest used RBA. See *OS/VS Checkpoint/Restart* for information about restarting the processing of VSAM data sets.

PROCESSING THE INDEX OF A KEY-SEQUENCED DATA SET

You can gain access to the index of a key-sequenced data set in one of two ways:

- By opening the cluster and using the GETIX and PUTIX macros
- By opening the index component alone and using the macros for normal data processing (GET, PUT, etc.)

Processing the index component alone is identical to processing an entry-sequenced data set—an index itself has no index and thus cannot be processed by keyed access. The *OS/VS Virtual Storage Access Method (VSAM) Programmer's Guide,* tells how to gain access to data records.

Processing an alternate index is identical to processing a key-sequenced data set; it too is described in the *VSAM Programmer's Guide.*

This chapter gives the format of VSAM index records, explains the meaning and use of the information in them, and tells how to gain access to them by way of GETIX and PUTIX. You should not attempt to duplicate or substitute for the index processing that VSAM does during normal access to data records.

The Format of an Index Record

Index records are stored in control intervals in the same way as data records, except that only one index record is stored in a control interval, and there is no free space between the record and the control information. Consequently, there is only one RDF, which contains the flag X'00' and the length of the record (a number equal to the length of the control interval minus 7). The CIDF also contains the length of the record (the displacement from the beginning of the control interval to the control information); its second number is 0 (no free space). The contents of the RDF and CIDF are the same for every used control interval in an index. The control interval after the last used control interval has a CIDF filled with 0s.

Index control intervals aren't grouped into control areas as are data control intervals. When a new index record is required, it is stored in a new control interval at the end of the index data set. As a result, the records of one index level aren't segregated from the records of another level, except when the sequence set is separate from the index set. Each index record is identified with respect to the level to which it belongs by a field in the index header—see the following section, "The Header Portion."

When an index record is replicated on a track, each copy of the record is identical to the other copies. Replication doesn't affect the contents of records.

Figure 4 sketches the parts of an index record.

The first part is a *header* that contains control information about the record.

Next, a sequence-set index record (a record in the lowest level of the index) may contain a set of *free-control-interval entries* used to locate free control

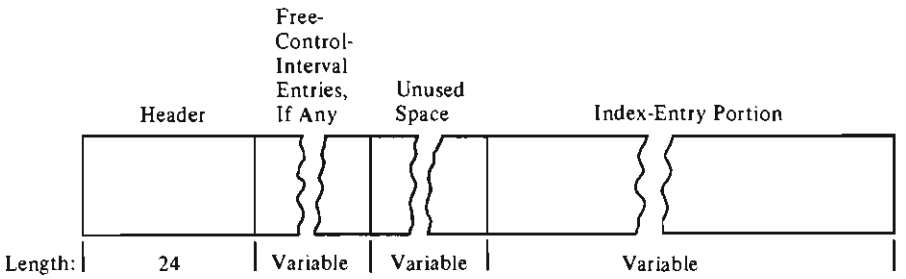


Figure 4. General format of an index record

intervals in the control area governed by the index record. An index-set index record (a record in a higher level of the index) has no entries of this type.

Next, an index record may have unused space.

The last part is a set of *index entries* used to locate, for an index-set record, control intervals in the next lower level of the index, or, for a sequence-set record, used control intervals in the control area governed by the index record.

The Header Portion

The first 24 bytes of an index record is the header, which gives control information about the index record. Figure 5 shows its format. All lengths and displacements are in bytes (binary). The discussions in the following two sections amplify the meaning and use of some of the fields in the header.

The Free-Control-Interval-Entry Portion

If the control area governed by a sequence-set record has free control intervals, the sequence-set record has entries pointing to those free control intervals. Each entry is 1, 2, or 3 bytes long (indicated by IXHPTLS in the header—the same length as the pointers in the index entries).

The entries come immediately after the header. They are used from *right to left*. The rightmost entry is immediately before the unused space (whose displacement is given in IXHFSO in the header). When a free control interval gets used, its free entry is zeroed out, the space becomes part of the unused space, and a new index entry is created in the position determined by ascending key sequence.

Thus, the free-control-interval-entry portion contracts to the left, and the index-entry portion expands to the left. When all of the free control intervals in a control area have gotten used, the sequence-set record governing the control area no longer has free-control-interval entries, and the number of index entries equals the number of control intervals in the control area.

Field	Offset	Length	Description
IXHLL	0(0)	2	Record length. The length of the index record is equal to the length of the control interval minus 7.
IXHFLPLN	2(2)	1	Index-entry control-information length. This is the length of the last three of the four fields in an index entry. (The length of the first field is variable.) The length of the control information is 3 or 4 in a sequence-set record and 5 in an index-set record.
IXHPTLS	3(3)	1	Vertical-pointer-length indicator. The fourth field in an index entry is a vertical pointer to a control interval. In an index-set record, the pointer is a binary number that designates a control interval in the index. The number is calculated by dividing the RBA of the control interval by the length of the control interval. To allow for a possibly large index, the pointer is always 3 bytes. In a sequence-set record, the pointer is a binary number, beginning at 0, that designates a control interval in the control area governed by the sequence-set record. A free-control-interval entry is nothing more than a vertical pointer. There are as many index and free-control-interval entries in a sequence-set record as there are control intervals in a control area. Depending on the number of control intervals in a control area, the pointer is 1, 2, or 3 bytes. An IXHPTLS value of X'01' indicates a 1-byte pointer; X'03' indicates a 2-byte pointer; X'07' indicates a 3-byte pointer.
IXHBRBA	4(4)	4	Base RBA. In an index-set record, this is the beginning RBA of the index. Its value is 0. The RBA of a control interval in the index is calculated by multiplying control-interval length times the vertical pointer and adding the result to the base RBA. In a sequence-set record, this is the RBA of the control area governed by the record. The RBA of a control interval in the control area is calculated by multiplying control-interval length times the vertical pointer and adding the result to the base RBA. Thus, the first control interval in a control area has the same RBA as the control area (length times 0, plus base RBA, equals base RBA).
IXHHP	8(8)	4	Horizontal-pointer RBA. This is the RBA of the next index record in the same level as this record. The next index record contains keys next in ascending sequence after the keys in this record.
	12(C)	4	(Reserved.)
IXHLV	16(10)	1	Level number. The sequence-set is the first level of an index, and each of its records has an IXHLV of 1. Records in the next higher level have a 2, and so on.
	17(11)	1	(Reserved.)
IXHFSO	18(12)	2	Displacement to the unused space in the record. In an index-set record, this is the length of the header (24)—there are no free-control-interval entries. In a sequence-set record, the displacement is equal to 24, plus the length of free-control-interval entries, if any.
IXHLEO	20(14)	2	Displacement to the control information in the last index entry. The last (leftmost) index entry contains the highest key in the index record. In a search, if the search-argument key is greater than the highest key in the preceding index record but less than or equal to the highest key in this index record, then this index record governs either the index records in the next lower level that have the range of the search-argument key or the control area in which a data record having the search-argument key is stored.
IXHSEO	22(16)	2	Displacement to the control information in the last (leftmost) index entry in the first (rightmost) section. Index entries are divided into sections to facilitate a quick search. Individual entries aren't examined until the right section is located.

Figure 5. Format of the header of an index record

The Index-Entry Portion

The index-entry portion of an index record takes up all of the record that is left over after the header, the free-control-interval entries, if any, and the unused space.

Figure 6 shows the format of the index-entry portion of an index record. To improve search speed, index entries are grouped into sections, of which there are approximately as many as the square root of the number of entries. For example, if there are 100 index entries in an index record, they are grouped into 10 sections of 10 entries each. (The number of sections doesn't change, even though the number of index entries increases as free control intervals get used.)

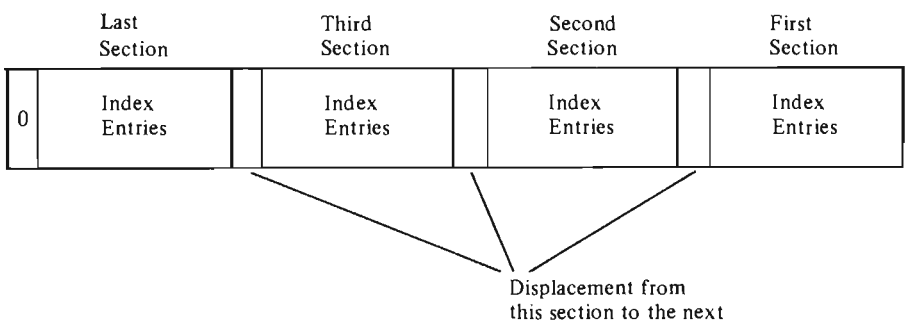


Figure 6. Format of the index-entry portion of an index record

The sections, and the entries within a section, are arranged from *right to left.* IXHLEO in the header gives the displacement from the beginning of the index record to the control information in the leftmost index entry. IXHSEO gives the displacement to the control information in the leftmost index entry in the rightmost section. You calculate the displacement of the control information of the rightmost index entry in the index record (the entry with the lowest key) by subtracting IXHFLPLN from IXHLL in the header—the length of the control information in an index entry from the length of the record.

Each section is preceded by a 2-byte field that gives the displacement from the control information in the leftmost index entry in the section to the control information in the leftmost index entry in the next section (to the left). The last (leftmost) section's 2-byte field contains 0s.

Figure 7 gives the format of an index entry. Index entries are variable in length within an index record, because VSAM compresses keys. That is, it eliminates redundant or unnecessary characters from the front and back of a key to save space. The number of characters that can be eliminated from a key depends on the relationship between that key and the preceding and following keys.

For *front compression,* VSAM compares a key in the index with the preceding key in the index and eliminates from the key those leading characters that are the same as the leading characters in the preceding key. For example, if key 12356 follows key 12345, the characters 123 are eliminated from 12356 because they are equal to the first three characters in the preceding key. The lowest key in an index record has no front compression—there is no preceding key in the index record.

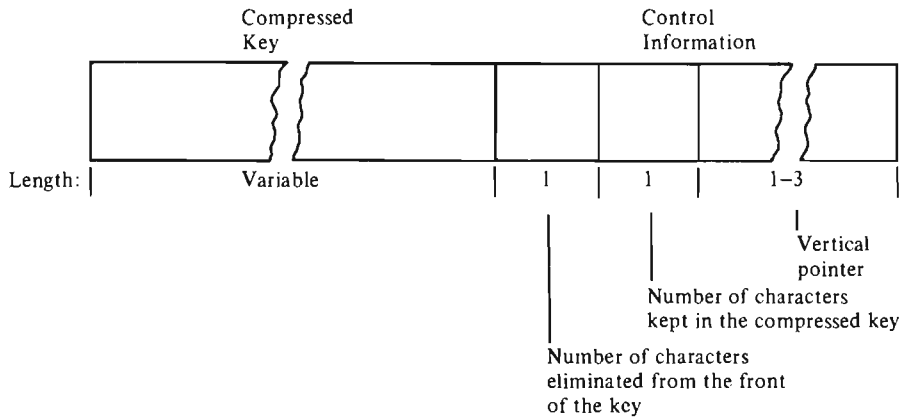


Figure 7. Format of an index entry

There is an exception for the highest key in a section. For front compression, it is compared with the highest key in the preceding section, rather than with the preceding key. The highest key in the rightmost section of an index record has no front compression—there is no preceding section in the index record.

For *rear compression,* VSAM compares a key in the index with the following key *in the data* and eliminates from the key those characters to the right of the first character that is unequal to the corresponding character in the following key. For example, if the key 12345 (in the index) precedes key 12356 (in the data), the character 5 is eliminated from 12345 because the fourth characters in the two keys are the first unequal pair.

The first of the control-information fields gives the number of characters eliminated from the front of the key, and the second field gives the number of characters that remain. When the sum of these two numbers is subtracted from the full key length (available from the catalog when the index is opened), the result is the number of characters eliminated from the rear. The third field indicates the control interval that contains a record with the key.

The example in Figure 8 gives a list of full keys and shows the contents of the index entries corresponding to the keys that get into the index (the highest key in each data control interval). A sequence-set record is assumed, with vertical pointers 1 byte long. The index entries shown in the figure from top to bottom are arranged from right to left in the assumed index record.

Key 12345 has no front compression because it is the first key in the index record. Key 12356 has no rear compression because, in the comparison between 12356 and 12357, there are no characters following 6, which is the first character that is unequal to the corresponding character in the following key.

You can always figure out what characters have been eliminated from the front of a key; you can't figure out the ones eliminated from the rear. Rear compression, in effect, establishes the key in the entry as a boundary value instead of an exact high key. That is, an entry doesn't give the exact value of the highest key in a control interval, but gives only enough of the key to distinguish it from the lowest key in the next control interval. In Figure 8, for example, the last three index keys, after rear compression, are 12401, 124, and 134. Data records with key fields between 12402 and 124FF are

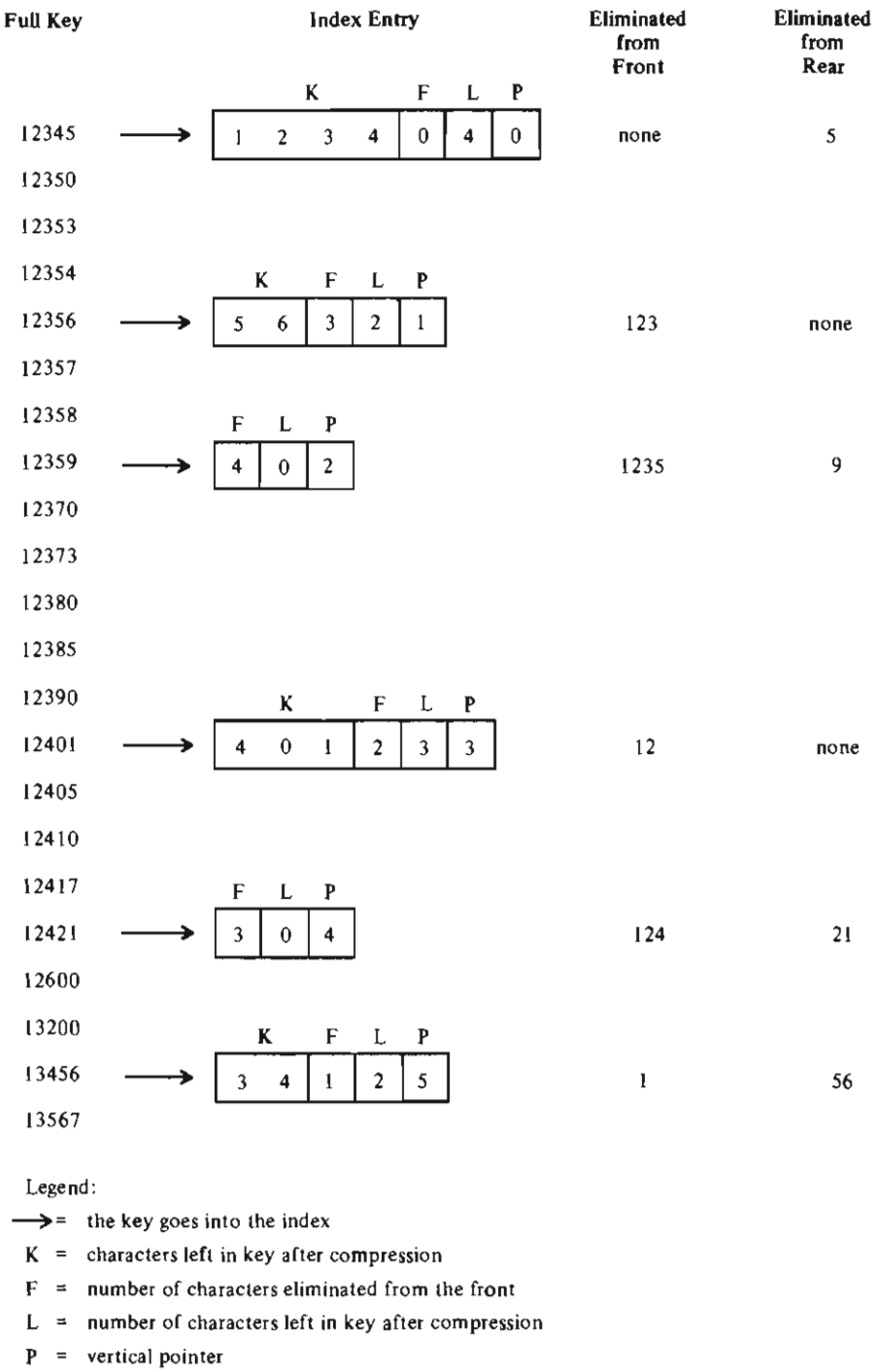


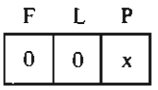
Figure 8. Example of key compression

associated with index key 124; data records with key fields between 12500 and 134FF are associated with index key 134.

If the data record with the highest key in a control interval is deleted, the index entry's key value doesn't change—doesn't change, in fact, even if all of the data records are deleted.

The last index entry in an index level indicates the highest possible key value. The convention for expressing this value is to give none of its characters and

indicate that no characters have been eliminated from the front. The last index entry in the last record in the sequence set looks like this:



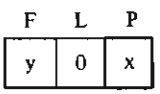
where x is a binary number from 0 to 255, assuming a 1-byte pointer.

In a search, the two 0s signify the highest possible key value in this way: The fact that 0 characters have been eliminated from the front implies that the first character in the key is greater than the first character in the preceding key. A length of 0 indicates that no characters need be compared to determine whether the search is successful—that is, when a search encounters the last index entry, a hit has been made.

Index Entries for a Spanned Record

In a key-sequenced data set, there is an index entry for each control interval that contains a segment of a spanned record. All of the index entries for a spanned record are grouped together in the same section. They are ordered from right to left according to the sequence of segments (first, second, third, and so on).

Only the last (leftmost) index entry for a spanned record contains the key of the record. The key is compressed according to the rules described above. All of the other index entries for the record look like this:



where y is a binary number equal to the length of the key (y indicates that the entire key has been "eliminated from the front"; L indicates that 0 characters remain) and x identifies the control interval that contains the segment.

When a spanned record is deleted, all of the index entries except the last (the one that contains the key) are deleted and free-control-interval entries are created to identify the freed control intervals.

How to Gain Access to a Key-Sequenced Data Set's Index

You can gain access to the index of a key-sequenced data set in one of two ways:

- By opening the cluster and using the GETIX and PUTIX macros
- By opening the index component alone and using the macros for normal data processing (GET, PUT, etc.)

Opening the Cluster

To process the index of a key-sequenced data set with GETIX and PUTIX, you must open the cluster with ACB MACRF=(CNV,...) specified. CNV provides for control-interval access, which you use to gain access to the index component.

GETIX and PUTIX are coded the same way as the other request macros—their only operand is RPL=address.

Access by way of GETIX and PUTIX is direct by control interval—VSAM requires RPL OPTCD=(CNV,DIR). The search argument for GETIX is the RBA of a control interval. The increment from the RBA of one control interval to that of the next is control-interval size for the index.

GETIX can be issued either for update or not for update. VSAM recognizes OPTCD=NUP or UPD but interprets OPTCD=NSP as NUP.

The contents of a control interval cannot be inserted by way of PUTIX—VSAM requires OPTCD=UPD. The contents must previously have been retrieved for update by way of GETIX.

RPL OPTCD=MVE or LOC may be specified for GETIX, but only OPTCD=MVE is valid for PUTIX. If you retrieve with OPTCD=LOC, you must change OPTCD to MVE to store. With OPTCD=MVE, AREALEN must be at least index control-interval size.

Beyond these restrictions, access to an index by way of GETIX and PUTIX follows the rules described in the chapter "Processing Control Intervals."

Error Return Codes

The error return codes for GETIX and PUTIX are the same as for GET and PUT, which are given in the *VSAM Programmer's Guide.*

Opening the Index Component of the Cluster

You can gain addressed or control-interval access to the index component of a key-sequenced cluster by opening the index component alone and using the request macros for normal data processing. To open the index component alone, specify:

DSNAME= *indexcomponentname*

in the DD statement identified in the ACB (or GENCB) macro.

You can gain access to index records with addressed access and to index control intervals with control-interval access. The use of these two types of access for processing an index is identical in every respect with their use for processing a data component.

PROCESSING A VSAM CATALOG

You can process a VSAM catalog as a catalog or as a data set. To process it as a catalog, you use Access Method Services commands and the VSAM SHOWCAT, SHOWCB, and TESTCB macros. To process it as a data set, you connect your processing program to it and use the VSAM request macros (GET, PUT, etc.).

This chapter describes gaining access to a catalog as a catalog and as a data set.

Gaining Access to a Catalog as a Catalog

Using Access Method Services to process a catalog is described in the appropriate *Access Method Services.* The catalog processing described there includes:

- Creating a catalog (DEFINE MASTERCATALOG and DEFINE USERCATALOG)
- Inserting records into a catalog (DEFINE)
- Modifying catalog records (ALTER)
- Deleting catalog records (DELETE)
- Listing catalog records (LISTCAT)

The SHOWCB and TESTCB macros enable you to retrieve information from a catalog by way of an access-method control block that connects your processing program to an open data set defined in the catalog. The *OS/VS Virtual Storage Access Method (VSAM) Programmer's Guide* describes the use of these macros.

Using the SHOWCAT Macro

The SHOWCAT (show, or display, the catalog) macro enables you to retrieve information from a catalog independently of an open data set defined in the catalog.

The entries in a catalog are interrelated. More than one entry is required to describe an object and its associated objects; one entry points to one or more other entries, which point to yet others. Figure 9 shows the interrelationship among entries that describe the following types of objects:

- Alternate index (G)
- Cluster (C)
- Data component (D)
- Index component (I)
- Path (R)
- Upgrade set (Y)

For example, an alternate-index entry points to the entries of its data and index components, its base cluster, and its path. SHOWCAT enables you to follow the arrows pictured in Figure 9. You first issue SHOWCAT on the name of an object. The information VSAM returns to you includes the

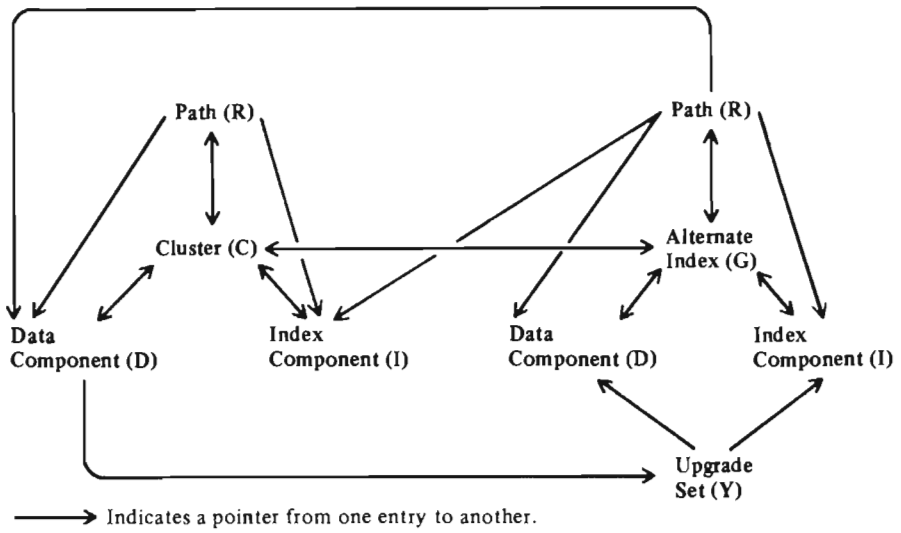


Figure 9. Interrelationship among catalog entries

control-interval numbers of catalog records in entries that describe associated objects. You then issue SHOWCAT on a control-interval number to retrieve information from one of these other entries.

The first time you issue SHOWCAT, VSAM searches VSAM catalogs in the following order to locate the entry that describes the object you name:

1. The STEPCAT or JOBCAT user catalog or catalogs (catalogs can be concatenated under STEPCAT or JOBCAT)
2. The master catalog
3. (VS2 only, when the object has a qualified name) the catalog, if any, whose name or alias is the same as the first-level qualifier of the object's name

VSAM returns to you the address of the access-method control block that defines the catalog. The subsequent times you issue SHOWCAT, you can specify that address causes VSAM to search only that catalog.

Format of the SHOWCAT Macro

The SHOWCAT macro has a standard form and list and execute forms. The standard form builds a parameter list and passes control to VSAM to display the catalog information. The list and execute forms are described in the appendix.

The format of the SHOWCAT macro is:

SHOWCAT	[**ACB**=*address*] **AREA**=*address* {**CI**=*address* | **NAME**=*address*}

ACB= *address*
specifies the address of the access-method control block that defines the catalog that contains the entry from which information is to be displayed. You issue the first SHOWCAT without ACB specified and VSAM supplies it to you for the next SHOWCAT (see the description of the work area under the AREA operand). Specifying ACB enables VSAM to go directly to the correct catalog without searching other catalogs first. You should always specify ACB when you specify CI instead of NAME.

AREA
specifies the address of the work area in which the catalog information is to be displayed. The first 2 bytes of the area must give the length of the area, including the 2 bytes. 64 is the minimum. If the area is too small, VSAM returns as much information as possible.

You can use the macro IGGSHWPL to generate a DSECT statement and labels for the fields in the work area.

The format of the area is:

Offset	Length	Description
0(0)	2	Length of the area, including the length of this field (provided by you).
2(2)	2	Length of the area actually used by VSAM, including the length of this field and the preceding field.
4(4)	4	The address of the ACB that defines the catalog that contains the entry from which information is displayed.
8(8)	1	Type of object about which information is returned: C Cluster D Data component G Alternate index I Index R Path Y Upgrade set The following fields contain one set of information for C, G, R, and Y types and another set for D and I types:
For C, G, R, and Y types:		
9(9)	1	For C and Y types: reserved. For G type: x... .... The alternate index might (1) or might not (0) be a member of an upgrade set. The way to find out for sure is to display information for the upgrade set of the base cluster and check whether it contains control-interval numbers of entries that describe the components of the alternate index. Figure 9 shows you how to get from the alternate index's catalog entry to the entries that describe its components (G to C to D to Y to D and I). .xxx xxxx Reserved. For R type: x... .... The path is (1) or isn't (0) defined for upgrading alternate indexes. .xxx xxxx Reserved.

Offset	Length	Description
10(A)	2	The number of pairs of fields that follow. Each pair of fields identifies another catalog entry that describes an object associated with this C, G, R, or Y object. The possible types of associated objects are: With C: D, G, I, R. With G: C, D, I, R. With R: C, D, G, I. With Y: D, I. Figure 9 shows how the catalog entries for all these objects are interrelated.
12(C)	1	Type of object the entry describes.
13(D)	3	The control-interval number of its first record.
16(10)		Next pair of fields, and so on. For type Y, if the area is too small to display a pair of fields for each associated object, VSAM displays as many pairs as possible and returns a code of 4 in register 15. For types C and G, if the area is too small, VSAM displays as many pairs as possible, but returns a code of 0 in register 15, because fields for the main associated objects can always be displayed (in the smallest allowed work area). For type R, fields for all associated objects (five possible) can always be displayed. (A pair of fields occupies 4 bytes, but, for all types except Y, 4 additional bytes are required as work space for the SHOWCAT processor. Thus, for example, if you provide 80 bytes for associated objects, as many as 10 pairs of fields can be displayed for type C or G and 20 for type Y.)
For D and I types:		
9(9)	1	Reserved.
10(A)	2	Relative position of the prime key in records in the data component. For the data component of an entry-sequenced or a relative record data set, there is no prime key, and this field is 0.
12(C)	2	Length of the prime key.
14(E)	4	Control-interval size of the data or index component.
18(12)	4	Maximum record size of the data or index component.
22(16)	2	The number of pairs of fields that follow. Each pair of fields identifies another catalog entry that describes an object associated with this D or I object. The possible types of associated objects are: With D: C, G, Y. With I: C, G. Figure 9 shows how the catalog entries for all these objects are interrelated.
24(18)	1	Type of object the entry describes.
25(19)	3	The control-interval number of its first record.
28(1C)		Next pair of fields, and so on. Fields for all associated objects can always be displayed.

{ **CI=** *address* | **NAME=** *address* }
specifies the address of an area that identifies the catalog entry that contains the desired information.

CI= *address*
specifies that the area is 3 bytes long and contains the control-interval number (RBA divided by 512) of the first record in the catalog entry. You can issue the first SHOWCAT with NAME specified and VSAM supplies control-interval numbers to you for other SHOWCATs (see the description of the work area under the AREA operand). The type of object named must be C, D, G, I, R, or Y. The 3-byte area must be

separate from the work area, even though VSAM returns a control-interval number in the work area.

NAME= *address*
specifies that the area is 44 bytes long and contains the name of the object described by the entry. The name is left-justified and padded with blanks. The type of object named must be C, D, G, I, or R.

Return Codes from SHOWCAT

VSAM returns a code in register 15 that indicates whether the SHOWCAT request was successful:

Reg. 15	Meaning
0(0)	VSAM completed the task.
4(4)	The area specified in the AREA operand is too small to display all pairs of fields for the associated objects.
8(8)	There is insufficient virtual storage to complete the task. (A GETMAIN failed.)
12(C)	Either the ACB address is invalid or the VSAM master catalog doesn't exist or isn't open.
16(10)	Tbe address specified in the AREA operand is outside the partition or address space of the program that issued SHOWCAT.
20(14)	The named object or control interval doesn't exist.
24(18)	There was an I/O error in gaining access to the catalog.
28(1C)	The control-interval number is invalid.
32(20)	The catalog record doesn't describe a C, D, G, I, R, or Y type of object.
36(24)	The interrelationship among catalog entries is in error. For example, an entry that was supposed to describe a certain type of object describes another type.
40(28)	There was an unexpected error code returned from catalog management to the SHOWCAT processor.

Gaining Access to a Catalog as a Data Set

A VSAM catalog is a key-sequenced data set, and gaining access to it is identical to gaining access to other key-sequenced data sets. The *VSAM Programmer's Guide* tells how to gain access to a key-sequenced data set by key and by RBA.

This section discusses the format of a VSAM catalog, how to open it, and how to process it.

The Format of a VSAM Catalog

A VSAM catalog is a key-sequenced data set with 44-byte keys and variable-length records. The arrangement and format of records in the data component are given in the *OS/VS1 Virtual Storage Access Method (VSAM) Logic* and in the appropriate *Catalog Management Logic*. The data component is divided into two key ranges: the low-address range and the high-address range. The index component is physically imbedded between the two ranges.

A control interval in the data component is 512 bytes long. Each control interval in the low-address range contains one record; each control interval in the high-address range may contain several records.

The key of a record in the high-address range is the name of the object the record describes (for example, a cluster, a path, an alternate index, a storage

volume)—hence, the records in the high-address range are called *true-name records.* The key of a record in the low-address range contains a byte of binary 0s, the 3-byte number of the control interval that contains the record, and 40 bytes of binary 0s and control information.

The first few records in the low-address range describe the catalog itself. The rest of the records in the low-address range are used for information associated with true-name records. Each true-name record contains the control-interval number of a record in the low-address range. There is no other information in a true-name record besides an object's name and the control-interval number of a record in the low-address range.

In some cases, more than one record in the low-address range is required to describe the object named in the true-name record. In these cases, the first record contains the control-interval number of the second, the second of the third, and so on—they are chained.

In yet other cases, there are objects associated with the object named in the true-name record. For example, a cluster has an associated data component and, perhaps, an associated index component, alternate index(es) or path(s). (Figure 9 shows the possible associations of all the different types of objects.) In these cases, the record or records in the low-address range that describe the object named in the true-name record contain the control-interval numbers of records that describe associated objects.

Opening a Catalog

To open a VSAM catalog as a data set, you specify it in the JCL for your job step, build an access-method control block to describe it, and issue the OPEN macro.

Specify the catalog in a DD statement whose ddname isn't a special system name. If it is a user catalog, also specify it in a JOBCAT or STEPCAT DD statement.

In the ACB that describes the catalog, indicate the DD statement (not JOBCAT or STEPCAT) in which the catalog is specified. If the catalog is password-protected, specify its master password.

Issue OPEN on the ACB that describes the catalog. In VS2, only a program that has been link-edited with authorization is allowed to open a VSAM catalog as a data set. (The authorized program facility is described in *OS/VS2 System Programming Library: Supervisor.*) If an unauthorized program tries to open a VSAM catalog, a code of 152—X'98'—is returned in the ERROR field of the ACB. (This is the same return code for failure to provide a password.)

Before proceeding to gain access to the catalog, ensure that its end-of-data set information agrees with its actual end-of-data-set, either by calling Access Method Services to process the VERIFY command or by issuing the VERIFY macro. For information about how to call Access Method Services and about the VERIFY command, see the appropriate *Access Method Services.* The VERIFY command opens the VSAM object being verified (in this case a catalog), issues the VERIFY macro, then closes the object.

Since you have already opened the catalog, you can issue the VERIFY macro yourself. Its format is:

VERIFY **RPL=** *address*

In the RPL, OPTCD=(CNV,...) must be specified. In the ACB used to open the catalog (or other VSAM object), MACRF=(CNV[,NCI],...) must be specified.

To process the catalog (or other VSAM object) sequentially after verifying it, you must first explicitly position for sequential processing with a POINT macro or a direct GET macro with RPL OPTCD=NSP.

Gaining Access to a Catalog

You can gain access to a catalog by key or by address. It is recommended that you limit your access to read-only retrieval, or retrieval and storage for update. Catalog records are interrelated, and insertion or deletion of records may invalidate the relationships between records. You should gain access to a catalog by way of Access Method Services for deleting and inserting catalog records.

You can retrieve a record in the high-address range of a catalog by key; use the name of the object whose data is desired for the search argument. You can retrieve a record in the low-address range by generic key or by RBA. You construct the first 4 bytes of the key of the record by adding a byte of binary 0s in front of the record's control-interval number. You calculate the RBA of the record by multiplying 512 times the record's control-interval number.

BUILDING PARAMETER LISTS FOR GENCB, MODCB, SHOWCB, AND TESTCB

The standard forms of GENCB, MODCB, SHOWCB, and TESTCB build a parameter list, put its address in register 1, and pass control to a VSAM routine to generate, modify, display, or test an access-method control block, exit list, or request parameter list. Other forms of the macros only build the parameter list (list forms) or only pass control to VSAM (execute forms). All of these forms are described in the *OS/VS Virtual Storage Access Method (VSAM) Programmer's Guide.*

You can avoid using a macro to build the parameter list by building it yourself and issuing the execute form of the macro to pass control to VSAM. This chapter explains how to build the parameter lists for GENCB, MODCB, SHOWCB, and TESTCB. The rules for combinations of codes in a parameter list are the same as the rules for combinations of operands in a macro, which are given in the *VSAM Programmer's Guide.*

You can avoid issuing the execute form of the macro by coding the linkage instructions that pass control directly to the VSAM control block manipulation routine. Before passing control, you must build the parameter list yourself.

The Format of the Parameter Lists

A parameter list for GENCB, MODCB, SHOWCB, or TESTCB is a list of fullword addresses. The first address points to a header entry that identifies the type of request and type of control block and gives other general information about the request. Each of the rest of the addresses in the parameter list points to an element entry that identifies the information you want to generate, modify, display, or test.

The fullwords in the parameter list must be contiguous, and the last one must have a 1 in its first bit. The header entry and each element entry may be separate from each other.

Figure 10 gives the formats of the header and element entries for the four request types.

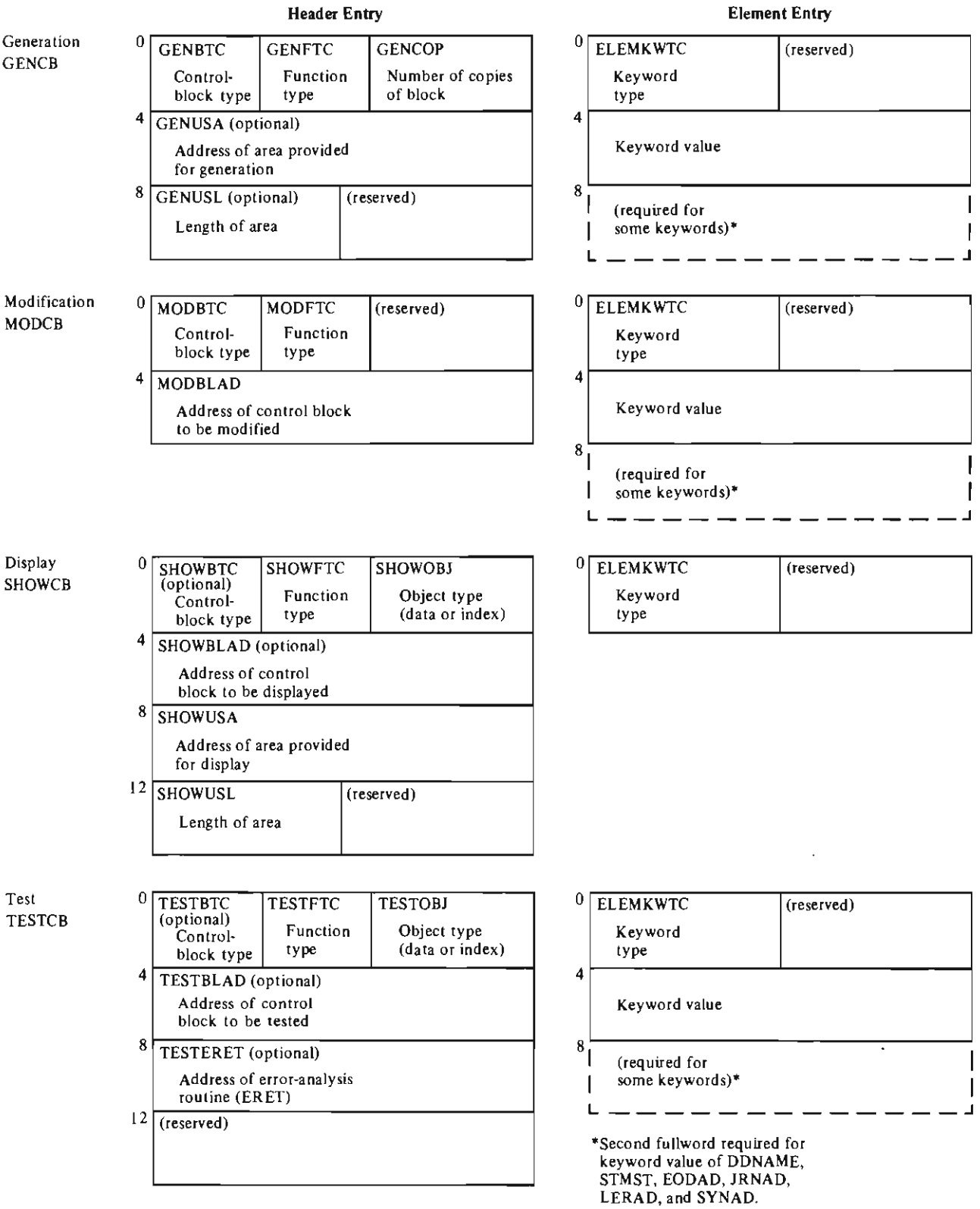


Figure 10. Format of header and element entries for GENCB, MODCB, SHOWCB, and TESTCB parameter lists

Building Header and Element Entries

Five assembler macros are provided for building entries. IDAGENC, IDAMODC, IDASHOW, and IDATEST help you build a header entry for generation, modification, display, or test. IDAELEM helps you build an element entry.

[*label*]	**IDAGENC**	**[DSECT={YES | NO}]**
[*label*]	**IDAMODC**	**[DSECT={YES | NO}]**
[*label*]	**IDASHOW**	**[DSECT={YES | NO}]**
[*label*]	**IDATEST**	**[DSECT={YES | NO}]**
[*label*]	**IDAELEM**	**[DSECT={YES | NO}]**

DSECT= { YES | NO }
This operand indicates whether a DSECT statement is to be generated. If you intend to build entries in a continuous area, you could have only the first of the macros generate a DSECT statement and use a single register to address the whole area.

These macros generate labeled DS statements that give the layout of an entry and EQU statements that pair a label with a numeric code. You can symbolically encode an entry with a series of move instructions. The macros are self-documenting—inspect a listing of their expansions and you can see what labels to code in your move instructions. (Either list the macros as they appear in the macro library, or code them and get an assembler listing.)

To generate an exit list with LERAD and SYNAD exits, you could code a GENCB of the standard form:

```
GENCB   BLK=EXLST,LERAD=(LOGERR,L),SYNAD=PHYSERR
```

The following example shows how to achieve the same effect by building the parameter list and entries yourself and issuing a GENCB of the execute form.

```
        LA      5,NTRYAREA          Set up base register for the entries.
        USING   5,GENC              GENC is the first label in the work
                                    area.
Build the list of addresses that point to the
entries.
        ST      5,PLIST             Address of the header entry.
        LA      6,GENLEN(,5)        Address of the first element entry.
                                    GENLEN is equated to the length of a
                                    header element for generation.
        ST      6,PLIST+4
        LA      6,ELEMLLEN(,6)      Address of the second element entry.
                                    ELEMLLEN is equated to the length of
                                    an element entry for an exit list.
        ST      6,PLIST+8
        OI      PLIST+8,X'80'       End-of-list indicator.
Build the header entry.
        MVI     GENBTC,GENXLST      Indicate the block type—exit list.
        MVI     GENFTC,GENFTYP      Indicate the function type—generation.
        MVI     GENCOP+1,X'01'      Indicate the number of copies of the
                                    exit list to be generated.
```

Build the first element entry.

```
         MVI     ELEMKWTC+1,      Indicate the keyword type—LERAD.
                 ELEMLEAD
         LA      6,LOGERR         Address of the name of the
                                  logical-error-analysis module.
         ST      6,ELEMPTR
         MVI     ELEMXFLG,        Indicate the presence of an address in
                 ELEMXL+ELEMXADR  ELEMPTR and that the exit routine is
                                  to be loaded.
```

Build the second element entry.

```
         LA      5,ELEMLLEN(,5)   Align the DSECT with the second
                                  element entry. ELEMLLEN is equated
                                  to the length of an element entry for an
                                  exit list.
         MVI     ELEMKWTC+1,      Indicate the keyword type—SYNAD.
                 ELEMSYAD
         LA      6,PHYSERR        Address of the entry point of the
                                  physical-error-analysis routine.
         ST      6,ELEMPTR
         MVI     ELEMXFLG,        Indicate the presence of an address in
                 ELEMXADR         ELEMPTR.
```

Pass control to VSAM.

```
         GENCB   MF=(E,PLIST)
         LTR     15,115           Generation successful?
         BNZ     CHECK0           No.
         .
         .
         .
CHECK0   ABEND   1,DUMP           Register 0 indicates the error.
```

Physical-error-analysis exit routine.

```
PHYSERR  ...
```

Work areas and constants.

```
LOGERR   DC      CL8'LEMOD'       Name of the logical-error-analysis
                                  module to be loaded.
PLIST    DC      3F'0'            List of entry addresses. 3 addresses are
                                  required: 1 for the header and 2 for the
                                  elements (1 for LERAD and 1 for
                                  SYNAD).
NTRYAREA DC      9F'0'            Work area for header and element
                                  entries. The header for GENCB is 3
                                  fullwords, and so are the LERAD and
                                  SYNAD elements.
```

DSECT with labels for the header and element entries.

```
         IDAGENC                  Header entry. A DSECT statement is
                                  generated, and register 5 is used to
                                  address NTRYAREA with these labels.
         IDAELEM DSECT=NO         Element entry. Element labels are part
                                  of the same DSECT as the header
                                  labels.
```

Passing Control Directly To VSAM

You can avoid using the execute form of GENCB, MODCB, SHOWCB, and TESTCB by building your own linkage instructions. You first build a parameter list, as described in the previous section, and put its address in register 1. Then you pass control to VSAM using the following instructions:

L	15,16	Put the address of the CVT into register 15.
L	15,256(,15)	Put the address of the AMCBS control block into register 15.
L	15,12(,15)	Put the address of the control block manipulation routine into register 15.
BALR	14,15	
	or	Branch to the routine
BAL	14,xx(,15)	

The BALR 14,15 instruction is used when the specific function (GENCB, MODCB, SHOWCB, or TESTCB) is not known, or when the control block type (ACB, EXLST, or RPL) is not known. The user-built parameter list contains the function code and control block type code.

The BAL 14,xx(,15) instruction is used to increase your program's performance. The "xx" is a decimal value that identifies a function (GENCB, MODCB, SHOWCB, or TESTCB) and a control block type (ACB, EXLST, or RPL.)

Decimal Value of xx	Function	Control Block
8	GENCB	ACB
12	GENCB	RPL
16	GENCB	EXLST
20	Reserved	
24	MODCB	ACB
28	MODCB	RPL
32	MODCB	EXLST
36	Reserved	
40	SHOWCB	ACB
44	SHOWCB	RPL
48	SHOWCB	EXLST
52	Reserved	
56	TESTCB	ACB
60	TESTCB	RPL
64	TESTCB	EXLST
68	Reserved	
72	SHOWCB or TESTCB	Blocklength keywords only
76[1]	SHOWCB	RECLEN field of an RPL
80[1]	MODCB	RECLEN field of an RPL

[1] Register 1 points to an RPL when "xx" is 76 or 80. See the following section for details.

When VSAM returns to your program, register 15 contains a completion code. Register 15 contains a zero value if the task completed successfully.

Otherwise, register 15 and register 0 contain codes that identify the reason VSAM couldn't complete the task. See "Error Return Codes" for details.

Modifying and Displaying the RECLEN Field of an RPL Directly

You can modify or display the RECLEN field (the record length) of an RPL without issuing a SHOWCB or MODCB macro, and without building a parameter list.

To modify the RECLEN field of an RPL, you first put the address of the RPL in register 1, and the value to be set in the RECLEN field in register 0. Next, you code the instructions that put the address of the VSAM control block manipulation routine into register 15, then branch to the routine:

```
L       15,16          Put the address of the CVT into register 15.
L       15,256(,15)    Put the address of the AMCBS control block into
                       register 15.
L       15,12(,15)     Put the address of the control block manipulation
                       routine into register 15.
BAL     14,80(,15)     Branch to the routine.
```

When VSAM returns to your program, register 15 contains a completion code. Register 15 contains a zero value if the field was modified correctly. Otherwise, register 15 and register 0 contain codes that identify the reason VSAM couldn't complete the task. See "Error Return Codes" for details.

To display the contents of the RECLEN field of an RPL, you first put the address of the RPL in register 1. Next, you code the instructions that put the address of the VSAM control block manipulation routine into register 15, and then branch to the routine:

```
L       15,16          Put the address of the CVT into register 15.
L       15,256(,15)    Put the address of the AMCBS control block into
                       register 15.
L       15,12(,15)     Put the address of the control block manipulation
                       routine into register 15.
BAL     14,76(,15)     Branch to the routine.
```

When VSAM returns to your program, register 15 contains a completion code. Register 15 contains a zero value if the field is displayed correctly, and register 0 contains the value of the RPL's RECLEN field. When register 15 is not zero, register 15 and register 0 contain codes that identify the reason VSAM couldn't complete the task. See "Error Return Codes" for details.

Error Return Codes

When you issue GENCB, MODCB, SHOWCB, or TESTCB, VSAM indicates whether it was able to perform the task by a code in register 15:

Reg. 15	Meaning
0(0)	VSAM completed the task.
4(4)	It couldn't complete the task.
8(8)	An attempt was made to use the execute form of a macro to modify a keyword that isn't coded in an element entry.

When register 15 contains 4, register 0 contains a code indicating the reason VSAM couldn't complete the task and register 1 contains the address of the element entry in error. In addition to the error return codes that are listed in the *VSAM Programmer's Guide*, you can get the following return codes in register 0 when you build your own parameter lists:

Reg. 0	Applicable Macros*	Meaning
1	G,M,S,T	The request type (whether to generate, modify, show, or test) is invalid.
2	G,M,S,T	The block type (access-method control block, exit list, or request parameter list) is invalid.
3	G,M,S,T	One of the keyword codes in the parameter list is invalid.
10	G,M	With GENCB, exit-list control-block type was specified and you specified an exit without giving an address. With MODCB, exit-list control-block type was specified and you specified an exit without giving an address; in this case, either active or not active must be specified, but load cannot be specified.
14	G,M,T	One of the option codes (for MACRF, ATRB, or OPTCD) has an invalid combination of option codes specified; for example, OPTCD=(ADR,SKP).
20	S	A keyword was specified that is meaningful only when MACRF=LSR or GSR was specified (but MACRF=LSR or GSR wasn't specified).
21	S,T	A keyword was specified that is available for displaying or testing only when the ACB does not represent a dummy data set.

*G=GENCB, M=MODCB, S=SHOWCB, T=TESTCB

APPENDIX: OPERAND NOTATION FOR THE MACROS

Among the macros presented in this publication, BLDVRP and SHOWCAT have list and execute forms, and DLVRP has an execute form. These are described in this appendix.

This appendix discusses operand notation for all of the macros presented in this publication. It gives a table that shows which expressions are applicable to each operand of these macros.

List and Execute Forms

The standard form of a VSAM macro builds a parameter list, puts its address in register 1, and passes control to VSAM to process it. Among the macros presented in this publication, BLDVRP and SHOWCAT have list and execute forms, and DLVRP has an execute form. The list form of a macro builds a parameter list when the macro is assembled—it is not executable. When the execute form of a macro is executed, the macro puts the address of the parameter list in register 1 and passes control to VSAM to process the list, and carry out the function specified in the list. An execute form may first change some of the values specified in the parameter list.

Using the list and execute forms of a macro enables you to code a reentrant program. You can share a parameter list among several tasks, so long as no task modifies it.

The BLDVRP Macro

The BLDVRP (build VSAM resource pool) macro has standard, list, and execute forms. The standard form is given in "Format of the BLDVRP Macro" in the chapter "Sharing Resources among Data Sets."

List Form of BLDVRP

The format of the list form of BLDVRP is:

BLDVRP	**BUFFERS=(***size* **(***number* **),***size* **(***number* **),...)** **MF=L** **[FIX={BFR | IOB | (BFR,IOB)}]** **[KEYLEN=** *length* **[STRNO=** *number* **]** **[TYPE={LSR | GSR}]**

MF=L
indicates that this is the list form of BLDVRP.

The other operands are described in the chapter "Sharing Resources among Data Sets." STRNO is optional in the list form of BLDVRP, but if it is not specified, it must be specified in the execute form.

Execute Form of BLDVRP

The format of the execute form of BLDVRP is:

BLDVRP	**MF=(E,** *address* **)** **[FIX={BFR | IOB | (BFR,IOB)}]** **[KEYLEN=** *length* **]** **[STRNO=** *number* **]** **[TYPE={LSR | GSR}]**

MF=(E, *address* **)**
indicates that this is the execute form of BLDVRP. The address is the address of the parameter list built by a list form of BLDVRP. If you use register notation, you may use register 1, as well as a register from 2 through 12. Register 1 is used to pass the parameter list to VSAM.

The other operands are described in the chapter "Sharing Resources among Data Sets." BUFFERS may not be specified in the execute form of BLDVRP because this operand affects the length of the parameter list.

The DLVRP Macro

The DLVRP (delete VSAM resource pool) macro has standard and execute forms. The standard form is given in "Format of the DLVRP Macro" in the chapter "Sharing Resources among Data Sets." There is no list form for DLVRP because DLVRP works with BLDVRP: it uses the parameter list associated with BLDVRP.

Execute Form of DLVRP

The format of the execute form of DLVRP is:

DLVRP	**MF=E** **TYPE={LSR | GSR}**

MF=E
indicates that this is the execute form of DLVRP.

The TYPE operand is described in the chapter "Sharing Resources among Data Sets."

The SHOWCAT Macro

The SHOWCAT (show, or display, the catalog) macro has standard, list, and execute forms. The standard form is given in "Using the SHOWCAT Macro" in the chapter "Processing a VSAM Catalog."

You can use the macro IGGSHWPL to generate a DSECT statement and labels for the fields in the parameter list for SHOWCAT.

List Form of SHOWCAT

The format of the list form of SHOWCAT is:

SHOWCAT	[**ACB**=*address*] [**AREA**=*address*] [{**CI**=*address* | **NAME**=*address*}] **MF=L**

MF=L
indicates that this is the list form of SHOWCAT.

The other operands are described in the chapter "Processing a VSAM Catalog." AREA and {CI | NAME} are optional in the list form of SHOWCAT, but if they are not specified, they must be specified in the execute form.

Execute form of SHOWCAT

The format of the execute form of SHOWCAT is:

SHOWCAT	[**ACB**=*address*] [**AREA**=*address*] [{**CI**=*address* | **NAME**=*address*}] **MF=({E | B},***address***)**

MF=({E | B},*address*)
indicates that this is the execute form of SHOWCAT.

E
indicates that the parameter list, whose address is given in *address,* is to be passed to VSAM for processing.

B
indicates that the parameter list is to be built or modified, but is *not* to be passed to VSAM. This form of the macro is similar to the list form, except that it works at execution time and can modify a parameter list, as well as build it.

To build a parameter list, first issue SHOWCAT with only MF=(B, *address*) specified, to zero out the area in which it will be built.

address
gives the address of the parameter list. If you use register notation, you may use register 1, as well as a register from 2 through 12. Register 1 is used to pass the parameter list to VSAM (MF=E).

The other operands are described in the chapter "Processing a VSAM CATALOG."

Expressions That Can Be Used for Operands

The values for an operand of the macros presented in this publication can be expressed in a variety of ways, as:

- An absolute numeric expression, for example, BUFFERS=(2048(10)).
- A code or a list of codes separated by commas and enclosed in parentheses, for example, FIX=BFR or FIX=(BFR,IOB).
- A register (in parentheses) from 2 through 12 that contains an address or numeric value, for example, BFRNO=(3). In the execute form of a macro, you can use register 1 for the address of the parameter list. Equated labels can be used to designate a register, for example, BFRNO=(BFR#), where the equate statement, BFR# EQU 3, has been included in the program.
- An expression valid for a relocatable A-type address constant, for example, AREA=RETURN+4.

The expressions that can be used depend on the operand. Only absolute numeric expressions, codes, and relocatable A-type address constants are valid for the list form of a macro.

A table follows that indicates the expressions allowed for each operand of the macros presented in this publication:

Operands	Absolute Numeric	Code	Register	A-Type Address
BLDVRP (Standard)				
BUFFERS	X			
FIX		X		
KEYLEN	X			
STRNO	X			
TYPE		X		
BLDVRP (List)				
BUFFERS	X			
FIX		X		
KEYLEN	X			
MF		X		
STRNO	X			
TYPE		X		
BLDVRP (Execute)				
FIX		X		
KEYLEN	X			
MF				
E		X		
address			X	X
STRNO	X			
TYPE		X		
DLVRP (Standard)				
TYPE		X		
DLVRP (Execute)				
MF		X		
TYPE		X		
GETIX				
RPL			X	X

Operands	Absolute Numeric	Code	Register	A-Type Address
MRKBFR				
MARK		X		
RPL			X	X
PUTIX				
RPL			X	X
SCHBRF				
BFRNO	X		X	
RPL			X	X
SHOWCAT (Standard)				
ACB			X	X
AREA			X	X
CI			X	X
NAME			X	X
SHOWCAT (List)				
ACB				X
AREA				X
CI				X
MF		X		
NAME				X
SHOWCAT (Execute)				
ACB			X	
AREA			X	
CI			X	
MF				
B		X		
E		X		
address			X	
NAME			X	
WRTBFR				
RPL			X	X
TYPE				
ALL		X		
CHK		X		
DS		X		
LRU		X		
percent	X		X	
TRN		X		

GLOSSARY

The following terms are defined as they are used in this book. If you do not find the term you are looking for, refer to the index or to the *IBM Data Processing Glossary,* GC20-1699.

base RBA: The RBA stored in the header of an index record that is used to calculate the RBAs of data or index control intervals governed by the index record.

CIDF: Control interval definition field. The 4-byte control-information field at the end of a control interval that gives the displacement from the beginning of the control interval to free space and the length of the free space. If the length is 0, the displacement is to the beginning of the control information.

control-interval access: The retrieval or storage of the contents of a control interval.

element entry: In a parameter list of GENCB, MODCB, SHOWCB, or TESTCB, an entry that identifies the keyword type and gives the information you want to generate, modify, display, or test.

free-control-interval entry: In a sequence-set index record, a vertical pointer that gives the location of a free control interval in the control area governed by the record.

front compression: The elimination, from the front of a key, of characters that are the same as the characters in the front of the preceding key.

GSR: Global shared resources (VS2 only). An option for sharing I/O buffers, I/O-related control blocks, and channel programs among VSAM data sets in a resource pool that serves all address spaces in the system.

header, index-record: In an index record, the 24-byte field at the beginning of the record that contains control information about the record.

header entry: In a parameter list of GENCB, MODCB, SHOWCB, or TESTCB, the entry that identifies the type of request and control block and gives other general information about the request.

horizontal pointer: In the header of an index record, the RBA of the index record in the same level as this one that contains keys next in ascending sequence after the keys in this one.

index entry: In an index record, a 4-field entry that contains the highest key stored in a control interval in the next lower index level or in a data control interval, the number of characters eliminated from the front of the key, the number of characters remaining in the key, and a vertical pointer to the control interval.

index level: A set of index records that order and give the location of all of the control intervals in the next lower level or in the data set that it controls.

index-record header: In an index record, the 24-byte field at the beginning of the record that contains control information about the record.

index set: The set of index levels above the sequence set. The index set and the sequence set together comprise the index.

key compression: The elimination of characters from the front and the back of a key that VSAM does not need to distinguish the key from the preceding or following key in the index record; reduces storage space for an index.

level number: For the index of a key-sequenced data set, a binary number in the header of an index record that indicates the index level to which the record belongs.

LSR: Local shared resources. An option for sharing I/O buffers, I/O-related control blocks, and channel programs among VSAM data sets in a resource pool that serves one partition or address space.

parameter list: Used by GENCB, MODCB, SHOWCB, and TESTCB, a list of addresses and entries that define the request and give the information required to carry it out. (*See also* header entry and element entry.)

RBA: Relative byte address. The displacement of a record or a control interval from the beginning of the data set to which it belongs; independent of the manner in which the data set is stored.

RDF: Record definition field. A 3-byte control-information field to the left of the CIDF that gives the length of a record in the control interval or the number of consecutive records having the same length.

rear compression: The elimination, from a key, of characters to the right of the first character that is unequal to the corresponding character in the following key.

resource pool, VSAM: (*See* VSAM resource pool.)

section: In an index record, a group of consecutive index entries. The index entries in an index record are divided into approximately as many sections as the square root of the number of entries, to speed up a search for an entry.

sequence set: The lowest level of the index of a key-sequenced data set; it gives the location of the control intervals in the data set and orders them by the key sequence of the data records they contain. The sequence set and the index set together comprise the index.

transaction ID: A number associated with each of several request parameter lists that define requests belonging to the same data transaction.

update number: For a spanned record, a binary number in the second RDF of a record segment that indicates how many times the segment has been updated. The update numbers of all the segments of a spanned record should be equal. An inequality indicates a possible error.

user buffering: The use of a work area in the processing program's address space for an I/O buffer; VSAM transmits the contents of a control-interval between the work area and direct-access storage without intermediary buffering.

vertical pointer: A pointer in an index record of a given level that gives the location of an index record in the next lower level or the location of a control interval in the data set controlled by the index.

VSAM resource pool: A virtual-storage area that is used to share I/O buffers, I/O-related control blocks, and channel programs among VSAM data sets. A resource pool is local or global (VS2 only); that is, it serves tasks in one partition or address space or tasks in all address spaces in the system.

INDEX

For additional information about any subject listed in this index, refer to the publications that are listed under the same subject in either *OS/VS1 Master Index*, GC24-5104, or *OS/VS2 Master Index*, GC28-0693.

A

ACB macro
  deferring writes from a buffer pool 27
  processing an index 46
  using a resource pool 25
  using control-interval access 17-18
access to
  catalog as a catalog 47-51
  catalog as a data set 51-53
  contents of control intervals 17-19,33
  index records 46
Access Method Services
  processing a catalog as a catalog 47
  VERIFY command 52
access-method control block
  displaying statistics about a buffer pool 23
  identifying a catalog with the SHOWCAT macro 49
address list, in parameter lists of GENCB, MODCB, SHOWCB, and TESTCB macros 55
addressed access
  intermixed with control-interval access 17
  to a catalog 53
  to an index 46
ALTER command
  processing a catalog as a catalog 47
alternate indexes
  processing as a data set 39
  providing buffers for shared resources 24
  upgrading when processing base cluster with control-interval access 33
authorized program facility
  fixing pages in real storage 19,24
  opening a VS2 catalog 52

B

base RBA, in index-record header
  how used to calculate the RBA of a control interval 41
  of control area 41
  of index 41
BLDVRP macro
  building a VSAM resources pool 23-25
  execute form 64
  list form 63
boundary values, of keys, through rear compression 43
buffer modification, journalling of 31-32
buffer pool
  connecting a data set to 25
  deferring writes from 27
  deleting 26
  managing 27-36
  providing 22-25
buffer, marking for output 35-36
buffer, releasing from control 35-36
buffering, I/O
  using work area as a buffer 18-19
  with shared resources 27-36
  (*see also* buffer pool)
building a resource pool 23-25
building parameter lists for GENCB, MODCB, SHOWCB, and TESTCB macros
  coding example 57-58
  format of parameter lists 55-56
  macros used 57

C

cancelling I/O-error processing in a JRNAD exit routine 31-32
catalog entries, relationship among 47-48,51-52
catalog, VSAM
  format of 51-52
  order of search with SHOWCAT macro 48
  processing as a catalog 47-51
  processing as a data set 51-53
CBIC (Control Blocks in Common) 20
chaining request parameter lists
  control-interval access 17
  not allowed with
    improved control-interval access 19
    MRKBFR macro 36
    SCHBFR macro 34
    WRTBFR macro 29 CHECK macro
  with control-interval access 18
  with the WRTBFR macro 29
checkpoint routine
  restriction with a global resource pool 37
  writing buffers 29
CIDF (control information definition field)
  end of data 13-14
  format 13-14
  illustration 13,15,16
closing a data set
  writing buffers 29
codes
  control information in control interval 13-14,39
  index record 41-42,44-45
  JRNAD exit parameter list 31-32
  parameter lists of GENCB, MODCB, SHOWCB, and TESTCB macros 57
  (*see also* error return codes)
coding example, of building parameter lists for GENCB, MODCB, SHOWCB, and TESTCB macros 57-58
common area of virtual storage 22,25
compression, key
  control information in index entry 42-45
  examples 42-45
  front 42-43
  highest possible key value 44-45
  illustration 44
  rear 43
    boundary values 43
control area
  base RBA of, in a sequence-set index record 41
  none in an index 39
control information
  control interval
    CIDF 13-14
    RDF 14-16
    responsibility for maintaining, with control-interval access 17

control information (continued)
index record
free-control-interval entry 40
header 40-41
index entry 42-45
section displacement field 42
parameter lists of GENCB, MODCB, SHOWCB, and TESTCB macros
address list 55
element entry 55-56
header entry 55-56
control interval
control information
CIDF 13-14
RDF 14-16
responsibility for maintaining, with control-interval access 17
illustration 13,15,16
in a catalog 51
in a key-sequenced data set
free, governed by free-control-interval entry in index 40
used, governed by index entry in index 40-41
in an index 39
password 17
control interval definition field (CIDF)
end of data 13-14
format 13-14
illustration 13,15,16
control-area splits, journalling of 31-32
control-interval access
improved 18-19
debugging with normal control-interval access 19
passwords 17
responsibility for maintaining control information 17
to a data set 17-19,33
to an index 46

D

data records, position in a control interval 14-16
deadlock in exclusive control, preventing 32
deferring write requests 27-28
DEFINE command
processing a catalog as a catalog 47
DELETE command
processing a catalog as a catalog 47
deleting a resource pool 26-27
direct access, to control intervals 17
displaying information about an unopened data set (SHOWCAT macro) 47-51
displaying statistics about a buffer pool 23
displaying transaction ID of request parameter list 28
distributed free space 40
DLVRP macro
deleting a resource pool 26-27
execute form 64
DSNAME parameter, in JCL, for specifying an index 46

E

element entry, in parameter lists of GENCY, MODCB, SHOWCB, and TESTCB macros 55-56
illustration 56
using macro to build 57-58
coding example 57-58
elimination of characters from key (*see* key compression)
end of data
indicated by CIDF 13-14
verifying 52-53
ENDREQ macro
giving up exclusive control 32
like MRKBFR macro 35
used with WRTBFR macro 29
entry
catalog 47-48,51-52
element, in parameter lists of GENCB, MODCB, SHOWCB, and TESTCB macros 55-56
free-control-interval, in index record 40
header, in parameter lists of GENCB, MODCB, SHOWCB, and TESTCB macros 55-56
index 41-45
entry-sequenced data set
loading with control-interval access 18
processing with shared resources 33
RDFs 14-16
EODAD exit with control-interval access 17
ERASE macro, not used with control-interval access 18
error return codes
from BLDVRP macro 25
from DLVRP macro 27
from GENCB, MODCB, SHOWCB, and TESTCB macros, with user-built parameter lists 61
from GETIX macro 46
from MRKBFR macro 36
from OPEN macro
catalog 52
control-interval access 20
shared resources 26
from PUTIX macro 46
from request macros (GET, PUT, Etc.)
buffers not available 29
for control-interval access 20
physical errors
control-interval access 20
JRNAD exit 31-32
program check with global shared resources 25
from SCHBFR macro 35
from SHOWCAT macro 51
from SHOWCB macro 23,28
from TESTCB macro 28
from WRTBFR macro 30
event dispatching (improved control-interval access) 19
EXCEPTION exit with deferred writes 30
exclusive control
preventing deadlocks 32
releasing 35-36
execute forms of macros 63-65
expressions that can be used for operands 66-67

F

fixing pages in real storage
  with improved control-interval access 19
  with shared resources 24
format
  BLDVRP macro 24,63-64
  catalog 51-52
  control interval
    CIDF 13-14
    in an index 39-45
    RDF 14-16
  DLVRP macro 26,64
  GETIX macro 46
  index record
    free-control-interval entry 40
    header 40-41
    index entry 42-45
  MRKBFR macro 36
  parameter lists of GENCB, MODCB, SHOWCB, and TESTCB macros
    address list 55
    element entry 55-56
    header entry 55-56
  PUTIX macro 46
  SCHBFR macro 34
  SHOWCAT macro 48-51,65
  VERIFY macro 53
  WRTBFR macro 29-30
free-control-interval entry, in an index record 40
free space
  distributed, position in a control interval 40
  unused space in a spanned record 16
  unused space in an index record 40
front key compression 42-45
  examples 42-45
  (*see also* key compression)

G

gaining access to
  a catalog as a catalog 47-51
  a catalog as a data set 51-53
  contents of control intervals 17-19,33
  index records 46
GENCB macro
  execute form used with parameter lists build by user 57-58
  linking to VSAM directly 59
  preparing to open an index 46
  return codes, with user-built parameter lists 61
  specifying buffering 17-18
  specifying control-interval access 17
  specifying shared resources 22
generic key, used to process catalog 53
GET macro
  end of data 17
  shared resources 21
GETIX macro 46
global shared resources (GSR) 22-23,25
GSR (global shared resources) 22-23,25

H

header, index-record 40-41
header entry, in parameter lists of GENCB, MODCB, SHOWCB, and TESTCB macros 55-56
  illustration 56
  using macros to build 57-58
    coding example 57-58
high-address range of a catalog 51-52
horizontal pointer, in index-record header 41

I

I/O buffering
  using work area as a buffer with control-interval access 18-19
  with shared resources 27-36
  (*see also* buffer pool)
I/O errors, journaling of 31-32
I/O-related control blocks shared 21
IDAELEM macro 57
  coding example 58
IDAGENC macro 57
  coding example 58
IDAMODC macro 57
IDASHOW macro 57
IDATEST macro 57
improved control-interval access 18-19
  debugging with normal control-interval access 19
  not allowed with shared resources 33
index
  format of
    control intervals 39-40
    records 39-45
  opening 46
  processing 46
  replication of records, no effect on contents 39
  use of index entries for search by key 41,43
index entry 41-45
  for spanned records 45
  illustration 43
  sectioned to improve search speed 42
  use for search 41,43
index level
  index set 39,41
  indicator in index-record header 41
  sequence set 39-41
index record
  free-control-interval entry 40
  header 40-41
  index entry 41-45
  replication, no effect on contents 39
  unused space 40
index set 39,41
  no free-control-interval entries 40
index-record header 40-41

J

journaling
  buffer modifications 31-32
  control-area splits 31-32
  I/O errors 31-32
JRNAD exit
  not taken with improved control-interval access 19
  with shared resources 31-32

format (continued)

K

key, use of index entries for search 41,43
key compression
  control information in index entry 42-45
  examples 42-45
  front 42-43
  highest possible key value 44-45
  illustration 44
  rear 43
    boundary values 43
key-sequenced data set
  can't load with control-interval access 18
  index entries for spanned records 45
  RDFs 14-16
keyed access
  intermixed with control-interval access 17
  not used to gain access to an index 46

L

level, index
  index set 39,41
  indicator in index-record header 41
  sequence set 39-41
list, parameter
  of GENCB, MODCB, SHOWCB, and TESTCB
    macros 55-58
  of JRNAD exit 31-32
list forms of 63-65
LISTCAT command
  processing a catalog as a catalog 47
loading a data set
  not allowed with shared resources 21
  with control-interval access 18
local shared resources (LSR) 22
locate mode, of control-interval access
  with shared resources 34-35
  work area as a buffer (invalid) 18
  with VSAM buffering 17
locating an RBA in a buffer pool 33-35
low-address range of catalog 51-52
LSR (local shared resources) 22-23,25

M

macros
  list and execute forms 63-65
  expressions that can be used for operands 66-67
  (*see also* format and individual macros)
managing your own I/O buffers
  using shared resources 27-36
  using the work area as a buffer 17-18
marking a buffer for output 35-36
MODCB macro
  execute form used with parameter lists built by user 55
  linking to VSAM directly 59
  return codes, with user-built parameter lists 61
modifying buffers, journaling of 31-32
move mode, of control-interval access
  with shared resources 34-35
  work area as a buffer 17-18
MRKBFR macro 35-36

N

notation for macros, operand 63-67

O

OPEN macro
  opening a catalog 52
  opening a data for control-interval access 20
  opening a data set with shared resources 25
  opening a key-sequenced cluster to process and index 46
  opening an index component 46
operand notation for macros 63-67

P

pages, fixing in real storage
  with improved control-interval access 19
  with shared resources 24
parameter list
  of GENCB, MODCB, SHOWCB, and TESTCB
    macros 55-58
  of JRNAD exit 31-32
password
  for control-interval access 17
  for processing a catalog 52
performance
  deferring writes 27
  improved control-interval access 19
  sharing resources 21
  writing buffers 29
physical-error analysis
  in a JRNAD exit routine 31-32
  with control-interval access 20
  with deferred writes 30
physical-error message, RBA field in 20
placeholders with shared resources 21,23-25
POINT macro
  control-interval access 18
  end-of-data 17
  shared resources 21
pointer (*see* horizontal pointer and vertical pointer)
positioning with shared resources 21
preventing deadlock in exclusive control 32
processing a VSAM catalog
  as a catalog 47-51
  as a data set 51-53
program checks with global shared resources 25
providing a resource pool 22-27
PUT macro
  deferring writes for 27
  loading an entry-sequenced data set with control-interval
    access 18
  updating with control-interval access 18
  using MRKBFR macro instead, with shared resources 33
PUTIX macro 46

Q

qualified names and catalog search with SHOWCAT
    macro 48

format (continued)

R

random access (*see* direct access)
RBA (relative byte address)
  base, in index-record header
    how used to calculate the RBA of a control interval 41
    of control area 41
    of index 41
  search argument for
    catalog access 52-53
    direct control-interval access 17
    index access 46
    searching buffer pool for 33-34
RBA field
  in JRNAD parameter list 31-32
  in physical-error message 20
RDF (record definition field) 14-16
  illustration 15,16
rear key compression
  boundary values 43
  examples 43-44
  (*see also* key compression)
record
  catalog 51-52
  data, position in control interval 14-16
  index
    free-control-interval entry 40
    header 40-41
    index entry 41-45
    replication, no effect on contents 39
    unused space 40
    spanned 15-16
record definition field (RDF) 14-16
  illustration 15,16
register notation 14
relating deferred requests by transaction 28
relative byte address (RBA)
  search argument for
    catalog access 52-53
    direct control-interval access 17
    index access 46
  (*see also* base RBA and RBA field)
relative record data set
  can't load with control-interval access 18
  RDFs 14,16
releasing exclusive or shared control
  ENDREQ macro 32
  MRKBFR macro 35-36
replication, of index records, no effect on contents 39
request parameter lists
  chaining
    for control-interval access 17
    not allowed with
      improved control-interval access 19
      MRKBFR macro 36
      SCHBFR macro 34
      WRTBFR macro 29
  transaction IDs 28
resource pool
  connecting a data set to 25
  deferring writes 27
  deleting 26
  managing I/O buffers 27-36
  providing 22-25
restrictions
  catalog processing 52-53
  control-interval access 18-19
  for shared resources 37
  index processing 46
return codes
  from BLDVRP macro 25
  from DLVRP macro 27
  from GENCB, MODCB, SHOWCB, and TESTCB macros, with user-built parameter lists 61
  from GETIX macro 46
  from MRKBFR macro 36
  from OPEN macro
    catalog 52
    control-interval access 20
    shared resources 26
  from PUTIX macro 46
  from SCHBFR macro 35
  from SHOWCAT macro 51
  from SHOWCB macro 23,28
  from TESTCB macro 28
  from WRTBFR macro 30
RPL macro
  control-interval access 17-19
  MRKBFR macro 36
  SCHBFR macro 34
  transaction IDs 28
  WRTBFR macro 29

S

SCHBFR macro 33-35
search argument
  catalog access 53
  index access 46
  key, use of index entries 41,43
  RBA, for direct control-interval access 17
search catalog with SHOWCAT macro, order of 48
searching buffer pool for RBA 33-35
section, in an index record
  displacement field 42
  illustration 42
  index entry 42-43
  purpose (to improve search speed) 42
segment of spanned record 14-16
sequence set, level of an index 39-41
  free-control-interval entries 40
sequential access to control intervals 17-18
SHAREOPTION 4 (incompatible with deferring write requests) 27
sharing resources among data sets 21-37
  (*see also* resource pool)
SHOWCAT macro
  determining size of resource pool 22
  format
    execute 64
    list 64
    standard 49-51
  processing a catalog as a catalog 47-51
SHOWCB macro
  determining size of resource pool 23
  displaying transaction ID of request parameter list 28
  execute form used with parameter lists built by user 55
  linking to VSAM directly 59
  processing a catalog as a catalog 47
  return codes, with user-built parameter list 61

format (continued)
skip sequential access
intermixed with control-interval access 17
not used to gain access to an index 46
spanned records
illustration 16
index entries for 45
RDFs 14-16
segments 14-16
update number 14,16
splitting control areas, journaling of 31-32
SRB dispatching (improved control-interval access) 19
statistics for a resource pool 23
subpool 231
(for shared resources) 22
SYNAD exit with deferred writes 30

T

TESTCB macro
execute form used with parameter lists built by user 55
linking to VSAM directly 59
processing a catalog as a catalog 47
return codes, with user-built parameter lists 61
listing transaction ID of request parameter list 28
transaction ID
relating deferred requests 28
writing related requests 30
true-name records in catalog 52

U

unused space (with a spanned record) 16
update number (in spanned-record segments) 14,16
updating a data set, with control-interval access
using work area as a buffer 18
with shared resources 33,35
with VSAM buffering 18
upgrading alternate index when processing base cluster with control-interval access 33
user buffering
using work area as a buffer with control-interval access 18-19
with shared resources 27-36
(*see also* buffer pool)

V

VERIFY command 52
VERIFY macro 52-53
vertical pointer, in index
free-control-interval entry 40
how used to calculate the RBA of a control interval 41
index entry 42-43
length indicator, in index-record header 41
VSAM
I/O buffering 18
use in development of utility programs and system control programs 3,13

W

writing buffers after deferred writes 28-30
WRTBFR macro 28-30

123

2314 Direct Access Storage Facility, IBM 19

OS/VS Virtual Storage Access Method (VSAM) Options for Advanced Applications (File No. S370-30) Printed in U.S.A. GC26-3819-3

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

IBM Technical Newsletter

This Newsletter No.	GN26-0898
Date	September 30, 1977
Base Publication No.	GC26-3819-3
File No.	S370-30
Prerequisite Newsletters	None

OS/VS Virtual Storage Access Method (VSAM) Options for Advanced Applications

© Copyright IBM Corp. 1973, 1975, 1976

This technical newsletter, a part of Release 6 of OS/VS1 and Release 3.7 of OS/VS2, provides replacement pages for the subject publication. These replacement pages remain in effect for any subsequent releases unless specifically altered. Pages to be removed and inserted are:

Cover, edition notice
3, 4
7, 8
19-20.1 (20.1 added)
71, 72

Each technical change is marked by a vertical line to the left of the change.

Summary of Amendments

Control Blocks in Common (CBIC) option (MVS only), which allows a user to place the VSAM control blocks associated with a VSAM data set into the Common Service Area (CSA) of MVS.

Note: *Please file this cover letter at the back of the base publication to provide a record of changes.*

IBM Corporation, P.O. Box 50020, Programming Publishing, San Jose, California 95150

Printed in U.S.A.

OS/VS Virtual Storage Access Method (VSAM)
Options for Advanced Applications
GC26-3819-3

Reader's Comment Form

Your comments about this publication will help us to improve it for you. Comment in the space below, giving specific page and paragraph references whenever possible. All comments become the property of IBM.

Please do not use this form to ask technical questions about IBM systems and programs or to request copies of publications. Rather, direct such questions or requests to your local IBM representative.

If you would like a reply, please provide your name, job title, and business address (including ZIP code).

Fold on two lines, staple, and mail. No postage necessary if mailed in the U.S.A. (Elsewhere, any IBM representative will be happy to forward your comments.) Thank you for your cooperation.

GC26-3819-3

Fold and Staple

First Class Permit
Number 439
Palo Alto, California

Business Reply Mail

No postage necessary if mailed in the U.S.A.

Postage will be paid by:

IBM Corporation
General Products Division
Programming Publishing—Department J57
1501 California Avenue
Palo Alto, California 94304

Fold and Staple

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)